Sistemas Informáticos
Curso 2006-07

Vulnerabilidad en dispositivos Bluetooth

Alberto Macho González
Francisco Javier Pérez Escrivá

Dirigido por:
Prof. Luis Javier García Villalba
Dpto. Ingeniería del Software e Inteligencia Artificial - LSI
Grupo de Análisis, Seguridad y Sistemas (GASS)

Facultad de Informática
Universidad Complutense de Madrid

Índice General

Introducción

Hoy en día todo el mundo tiene un teléfono móvil, debido a la gran demanda y posibilidades prácticas, esta tecnología evoluciona a una gran velocidad. Conforme avanza el tiempo se está convirtiendo en una herramienta de trabajo necesaria, abriendo su abanico de funcionalidades.

Todas estas funciones hacen de un teléfono móvil un verdadero ordenador personal a disposición del usuario. Sin embargo, no todo son ventajas: a la vez que crece la capacidad de cómputo, crecen las posibilidades de instalar programas más potentes, con la capacidad de hacer cada vez más cosas, muchas de las cuales no van a ser siempre bienintencionadas; en otras palabras, el programador malintencionado tiene cada vez más medios a su disposición. Surge la necesidad entonces de realizar aplicaciones de seguridad para los dispositivos móviles y, en el caso que nos ocupa, para aquellos basados en el sistema operativo Symbian.

Symbian es el sistema operativo producto de la alianza de varias empresas de telefonía móvil, entre ellas Nokia, Sony Ericsson, Samsung y Siemens. El objetivo era competir con el sistema operativo de Palm o el Smartphone de Microsoft. El nombre de Symbian apareció por primera vez allá por 1998, aunque no fue hasta dos años más tarde que se comercializó el primer teléfono basado en Symbian, el Ericsson R380, con la versión 6 del sistema operativo. A partir de ahí diversos aparatos han sido puestos en el mercado, junto con las sucesivas versiones de Symbian, sobre todo del fabricante Nokia. Paralelamente se ha ido desarrollando el emulador EPOC para realizar programación en Symbian sobre Windows.

En el momento de la realización de este Proyecto, Symbian se encuentra en su versión 9.1 y existen dos grandes familias: por un lado las series S (40, 60, 80) de Nokia y por otro lado la plataforma UIQ utilizada por otras compañías como, por ejemplo, Sony Ericsson. Estas dos familias se diferencian sobre todo en la interfaz gráfica usada, implementada por encima de la capa del sistema operativo Symbian. Este proyecto se centrará solamente en los móviles de la serie 60.

Una de las utilidades que está en auge en un teléfono móvil es el uso de la tecnología Bluetooth, que permite una comunicación a corta distancia entre un teléfono móvil y otro dispositivo Bluetooth. Esta tecnología, de la cual hablaremos más adelante con mayor profundidad, ha abierto muchas posibilidades y al mismo tiempo muchas “inseguridades”, las cuales hay que tener en cuenta porque la información de un teléfono móvil puede ser privada o que en malas manos pueda ser utilizada de manera fraudulenta.

El objetivo de este proyecto es el estudio de las posibles vulnerabilidades que puede tener un teléfono móvil por medio de la tecnología Bluetooth, además

del estudio de viabilidad de una herramienta que ayude a vigilar el "medio" Bluetooth para evitar un mal uso del mismo.

Hoy en día hay herramientas que realizan esta labor en otro tipo de redes, son los husmeadores o sniffers. Este proyecto abordará el estudio de si se podría realizar un sniffer para Bluetooth sobre un dispositivo móvil que contenga un sistema operativo Symbian.

Introduction (*in English*)

Nowadays, everyone have a celular, and it is possible thanks to its great demand and its many and several applications, this technology is growing up so fast. As time goes, the celular becomes in an essential tool for many and different jobs, showing us how far we can go with it.

All these applications make the cellular a real personal computer at the service of the users. However, there are not only advantages: as the same time as the technological abilities increase, and the applications becomes in better ones, the intention is not always the best one.

In other words, the unkind software programmer has such a many ways to make whatever he wants. We saw the need to use security applications into the celulars, and specifically, into these ones based on the Symbian System.

Symbian is the operative system result of the alliance of several communications companies, like Nokia, Sony Ericsson, Samsung and Siemens. The objective to reach was to fight versus Palm´s operative system and the Microsoft´s Smartphone. The first time we saw the name of Symbian was in 1998, but the first cellular with this operative system wasn´t born until two years left, the Ericsson R380. This one had the V.6 of the Symbian System. From then, several celulars have been launched to the market, with the different version of Symbian System, especially Nokia ones. The EPOC simulator has been developed in parallel to program in Symbian on Windows.

At the moment, at the time when we was making this project, Symbian System is on his ninth version and exists two great and different families: on the one hand we have Nokia´s “S” series (40, 60, 80), but in the other, we have the UIQ platform used by companies like Sony Ericsson. These two families have a difference in the graphic interface used, that was built over Symbian operative system layer. This project will be only centered in the 60 series celulars.

One of the most used utilities nowadays, is the Bluetooth technology. This technology allows a short distance communication between a celular and another Bluetooth device. This use of Bluetooth technology, allows us so many possibilities and insecurity too. These insecurities are very important because the information and the data from a celular could be personal or in the wrong hands could be use badly.

The objective of the project is research about the bad use of the Bluetooth technology, and the research of the viability about any tool that help us to control the Bluetooth technology allowing us to check the right use of this technology.

At the moment, there are several tools that make this job in other kind of net (sniffers). This project will tackle the viability about the creation of a sniffer to the Bluetooth, always into the Symbian environment.

Objetivos

Antes de nada vamos a señalar los objetivos conseguidos de este proyecto al igual que las pautas para llegar a ellos.

- Estudio del sistema operativo Symbian para su uso a favor en nuestro proyecto, concretamente para móviles de la serie60.

- Estudio de la tecnología Bluetooth, y en concreto en dispositivos móviles.

- Estudio de las APIs que proporciona Symbian para el manejo de Bluetooth.

- Estudio de otras herramientas sniffer, para ver la viabilidad sobre el sistema operativo Symbian y sobre redes Bluetooth.

Sistema Operativo Symbian en el mundo

Symbian es un sistema operativo en la que coexisten dos grandes familias: por un lado las series S (40, 60, 80) de Nokia y por otro lado la plataforma UIQ utilizada por otras compañías como Sony Ericsson. Estas dos familias se diferencian sobre todo en la interfaz gráfica usada. Se trata de un sistema operativo libre, con gran cantidad de APIs que facilitan la programación sobre el mismo.

Existen numerosos modelos de terminales en el mercado que funcionan con Symbian. Cada plataforma tiene sus pros y sus contras. La plataforma Series 60 es la más común y la recomendada para comenzar con la programación en Symbian.

Este sistema operativo es similar a cualquier otro, como Windows, por ejemplo (de hecho también hay terminales con Windows).

Esto abre una cantidad ilimitada de posibilidades a través de la instalación de determinados programas, de las cuales enumero las más interesantes a continuación:

Ver películas en DivX: hay móviles sin Symbian que también permiten la reproducción de películas, pero entre la pequeña pantalla y que el archivo es un .3gp y no un .avi (con la merma de calidad que ello supone) pues esa posibilidad no suele contarse.

Lector de ebooks: Pues de lo más normal es descargarse algún libro y leerlo en el móvil. Obviamente los programas que realizan esta función disponen de zoom para no dejarnos la vista ahí.

Lector de noticias: No sabéis lo agradecido que es sincronizar el móvil por la mañana con tu PC y leer las noticias de los principales diarios del país desde

cualquier sitio y a cualquier hora en tu propio móvil, sin pagar nada, claro.

GPS: Una de las cosas que más le llama la atención a la gente es poder usar su móvil como GPS (con un receptor independiente por Bluetooth). Además de la guía puede decirte donde hay radares, y al no ser un antirradar no es ilegal.

Editor de imágenes: Cuando quieres mandar un MMS (Mensaje multimedia) y quieres personalizarlo con algún retoque fotográfico.

Mando a distancia de la televisión: Si tu móvil Symbian tiene puerto de infrarrojos puedes hacer que sirva como mando de una gran cantidad de mini cadenas, TV, DVD`s.

Videojuegos: Obviamente, el 90% de los terminales de hoy en día tienen juegos, pero no son Symbian. Los juegos de esta plataforma son mucho mejores, con mejor sonido, mejores gráficos… mención aparte para la Nokia n-cage el Symbian por excelencia para juegos, con el potencial de la primera PlayStation de Sony.

Edición de documentos: Se puede manejar archivos word, excel, powerpoint, y la mayoría también se pueden editar en el propio móvil.

Messenger: Desde cualquier parte se puede chatear con MSN Messenger.

Control de emule: Se puede ver información de tu programa de descarga emule desde cualquier lugar y controlar como van las descargas.

Empezar con Symbian

Así pues el primer paso a la hora de enfrentarse a Symbian consiste en la elección de la plataforma y la instalación del correspondiente emulador. Teniendo en cuenta que una premisa era utilizar la versión más reciente del sistema

operativo, dicha elección se reducía a las Series 60 y UIQ, en sus terceras versiones cada uno. Una vez probados los dos emuladores con pequeños programas de prueba no constaté ninguna diferencia importante así que me ceñí a las recomendaciones y decidí usar el emulador de las **Series 60 3rd edition** con **Symbian 9.1**.

Una vez elegida la plataforma, el siguiente paso del desarrollador es elegir un entorno de programación. Para ello existen diversas herramientas a su disposición, tanto de pago como de software libre. Lo primero a determinar es el lenguaje de programación que se quiere usar ya que Symbian 9.1 permite el desarrollo tanto en **C++** nativo como en **Java**. En este punto se optó claramente por **C++** ya que es el lenguaje más próximo del sistema operativo y porque Java posee muchas limitaciones, sobre todo en lo que se refiere al acceso a los recursos del teléfono móvil.

Tras diferentes pruebas con cada herramienta disponible para desarrollar en C++, se extrajeron unas conclusiones, plasmadas en el cuadro siguiente.

Herramienta	Distribución	Conclusiones
Metrowerks Codewarrior	Shareware (hay que pagar licencia al cabo de X días)	+ Realización automática de los procesos de compilación y enlazado. + Gran cantidad de documentación ya que es la herramienta más usada. - Periodo de prueba insuficiente - Interfaz bastante compleja
Carbide.c++	Libre	+ Entorno muy agradable y manejable (basado en Eclipse). + Tiene auto completar para el código. + Contiene plantillas de programas tipo. + Proceso de compilación y enlazado muy sencillo - Poca documentación

La herramienta usada finalmente fue **Carbide.c++** que, aunque se echa en falta algo más de documentación y tutoriales, es la más sencilla y agradable de usar, a parte de ser completamente gratuita.

Fundamentos de Symbian

Una vez hecha la elección de la herramienta a usar, conviene conocer ciertos aspectos sobre Symbian antes de ponerse directamente a programar.

Cada entorno de desarrollo organiza la información de forma parecida pero distinta, con lo cual no se puede editar directamente una aplicación hecha en otro entorno; para ello existen opciones de importación de proyectos que realizan la conversión de un entorno a otro (que no siempre se realiza satisfactoriamente). En la herramienta utilizada, Carbide.c++, el proyecto se organiza en diferentes directorios y archivos cuyo contenido se explicará a continuación. Antes de nada, hay que saber que las herramientas originales son de línea de comandos y que los entornos de programación únicamente facilitan el proceso de compilación y enlazado haciendo transparente el uso de estas herramientas.

El emulador EPOC

Paralelamente al crecimiento de Symbian, se ha ido desarrollando el emulador EPOC, capaz de imitar el comportamiento del teléfono móvil en un PC Windows. Son numerosos los diferentes emuladores disponibles en el mercado, cada uno de ellos adaptado a un modelo de móvil y a su versión de Symbian correspondiente. Para la realización del presente proyecto se eligió el emulador de las Series 60 de Nokia en su tercera edición por razones de comodidad, habiendo probado antes otros emuladores como, por ejemplo, el de UIQ. Una vez instalado el emulador de Symbian seleccionado (en este caso el *Series 60 3rd edition*), el sistema operativo (Windows) se configura con las correspondientes variables de entorno para ejecutarlo.

Nótese que las variables de entorno corresponderán al último emulador instalado. En este momento basta con ejecutar desde la línea de comandos:

Consola > **epoc**

Y se lanzará el emulador. Una vez dentro tendremos acceso a la mayor parte de la funcionalidad que posee el móvil, con algunas limitaciones claras (no se pueden efectuar llamadas por ejemplo). El emulador contiene varias opciones de configuración como puede ser la resolución de la pantalla; para ello remito al lector a la documentación sobre el emulador utilizado.

El directorio de instalación típico del emulador es C:\Symbian\9.1\S60_3rd.

Dentro de este directorio tenemos varios directorios interesantes que hay que tener en cuenta, que contienen el siguiente material:

- Documentación del emulador:
  \S60Doc

- Ejemplos de programas en Symbian nativo y específicos de la plataforma S60 respectivamente:

  \Examples
  \S60Ex

- Herramientas diversas para Symbian:

  \S60Tools

- El emulador en sí:

  \Epoc32

Que a su vez contiene los siguientes directorios relevantes, a parte de muchos más:

- Sistema de ficheros del móvil (discos **c:** , **d:** y **z:** del móvil respectivamente):

  \Epoc32\winscw\c
  \Epoc32\winscw\d
  \Epoc32\release\winscw\udeb\z

- Librerías de Symbian (para incluir en proyectos nuevos):

  \Epoc32\release\winscw\udeb

El sistema de ficheros es análogo al presentado por los teléfonos móviles y consta de dos discos en **RAM** (c y d) y un disco en **ROM** (z), de sólo lectura, en el que se almacena el kernel del sistema operativo de forma segura.

Herramientas de línea de comandos

Una vez instalador el emulador de Symbian en el PC, se instalan a su vez el compilador de C++ y las siguientes herramientas de línea de comandos:

- la herramienta **bldmake** que realiza una compilación de los ficheros fuente y los prepara para su construcción. Genera un ejecutable abld.bat que se utilizará en el siguiente paso. Se ejecuta de la siguiente forma:

  Consola > **bldmake bldfiles**

- la herramienta **abld** que realiza la construcción de la aplicación para una determinada plataforma; las más comunes son **winscw**, que es el emulador de Symbian, y **armv5**, que es el dispositivo móvil en sí. Así pues no es lo mismo construir la aplicación para el emulador que para el móvil y se pueden encontrar problemas al migrar una aplicación de emulador a móvil y viceversa.

  Consola > **abld build [winscw | armv5] [udeb | urel]**

En cuanto a la opción final se refiere, **udeb** crea los binarios con información simbólica para realizar depuración mientras que **urel** los crea sin esta información de depuración para usarlos como versión de salida (binarios más pequeños).

Para usar correctamente estas herramientas, el proyecto debe ser organizado correctamente y deben existir unos archivos que le indiquen a las herramientas lo que deben hacer:

- Por un lado tenemos los **ficheros fuente**: ficheros de cabecera **.h** con sus correspondientes ficheros de implementación **.cpp**.
- Debe existir un fichero **.mmp** que incluya información tal que el nombre final del ejecutable, ficheros fuente a compilar, librerías incluidas y permisos de la aplicación. Podemos ver el ejemplo siguiente, extraído del programa de ejemplo de uso de buffers y descriptores incluido en el e-emulador *S60 3rd edition*:

```
// DynamicBuffers.mmp

// Copyright (C) Symbian Software Ltd 2000-2005. All rights reserved.
```

```
// using relative paths for source and userinclude directories

// No explicit capabilities required to run this.

TARGET DynamicBuffers.exe
TARGETTYPE exe
UID 0
VENDORID 0x70000001
SOURCEPATH .
SOURCE DynamicBuffers.cpp
USERINCLUDE .
USERINCLUDE ..\..\CommonFramework
SYSTEMINCLUDE \Epoc32\include
LIBRARY euser.lib
CAPABILITY None
```

- Debe existir además un fichero **.inf** con información sobre la construcción final del programa. En el ejemplo descrito contiene únicamente una referencia al fichero .mmp.

```
// BLD.INF
// Component description file
//
// Copyright (C) Symbian Software Ltd 2000-2005. All rights reserved.
PRJ_MMPFILES
DynamicBuffers.mmp
```

Para más información, se recomienda acudir a la documentación de Symbian y, sobre todo, observar y probar los programas de prueba que se incluyen con el emulador.

Sin embargo, como ya se ha dicho anteriormente, el uso de un entorno de programación facilita este proceso enormemente, haciendo que todo sea transparente al programador.

Más adelante se describirá cómo trabaja la herramienta Carbide.c++ y cómo estructura la aplicación.

Proceso de construcción en Carbide.c++

Una vez ejecutado Carbide.c++, tenemos dos opciones: importar un proyecto existente a través de su archivo **.mmp** o crear un proyecto nuevo eligiendo una de las plantillas disponibles. En ambos casos, la herramienta creará un directorio de proyecto con, a su vez, varios directorios que contendrán los diferentes archivos utilizados. De estos directorios los más importantes, del punto de vista del programador, son:

- NombreProyecto

- NombreProyecto\.generated

- NombreProyecto\.settings

- **NombreProyecto\data**

- NombreProyecto\gfx

- **NombreProyecto\inc**

- NombreProyecto\S60 3.0 Emulator Debug

- NombreProyecto\sis

- **NombreProyecto\src**

Contienen, respectivamente, los archivos de recurso (**data**), los archivos de cabecera (**inc**) y los archivos fuente (**src**).

Los archivos de recurso contienen código que representan elementos de la interfaz gráfica. Veremos algún ejemplo en posteriores apartados.

En el caso de crear un proyecto nuevo, deberemos elegir qué tipo de aplicación nos interesa. Fundamentalmente son los siguientes tipos:

- **Librería dinámica (DLL):** si queremos crear un proyecto que simplemente ofrezca una funcionalidad en forma de librería dinámica.

- **Librería estática (LIB):** si queremos crear un proyecto que simplemente ofrezca una funcionalidad en forma de librería estática.

- **Aplicación básica de consola**: útil si sólo queremos hacer un programa que funcione sin necesidad de interfaz gráfica. No funciona en todos los emuladores.

- **Aplicación de GUI**: aplicación con interfaz gráfica.

Habrá que seleccionar también para qué objetivo queremos construir la aplicación. En el caso de que tengamos varios emuladores instalados (por ejemplo uno de Series 60 y uno de UIQ), Carbide.c++ nos dejará elegir entre ellos.

Una vez hecho esto, la herramienta nos generará los archivos necesarios, estructurados según los directorios citados anteriormente. Para construir la

aplicación bastará con hacer clic en Project / **Build All**; para lanzar el emulador hay que hacer clic en **Run**. Una vez dentro del emulador hay que encontrar el ejecutable del nuevo programa y lanzarlo; en el caso del *Series 60 3rd edition* hay que acceder al directorio Installat, en el cual se encontrará la aplicación recientemente creada.

Un aspecto importante a la hora de la compilación es la inclusión de **librerías** utilizadas para el proceso de enlazado. En Carbide.c++ hay que realizar dos pasos muy sencillos: primero se debe incluir el archivo de cabecera de la librería utilizada en nuestro código; acto seguido hay que acceder a las propiedades del proyecto y en el apartado *C/C++ Build*, en *WINSCW C/C++ Linker* y por último *Libraries* se debe hacer clic en *Add* y recorrer los directorios hasta encontrar la librería deseada (archivo .lib).

Aplicación de interfaz gráfica

Representa el tipo de aplicación más usado por parte de los desarrolladores. En este apartado se intentará explicar, desde un punto de vista global, cómo funciona la interfaz gráfica en Symbian y las pautas que hay que tener en cuenta a la hora de utilizarla. Para más detalles se remite al lector a la documentación de Symbian.

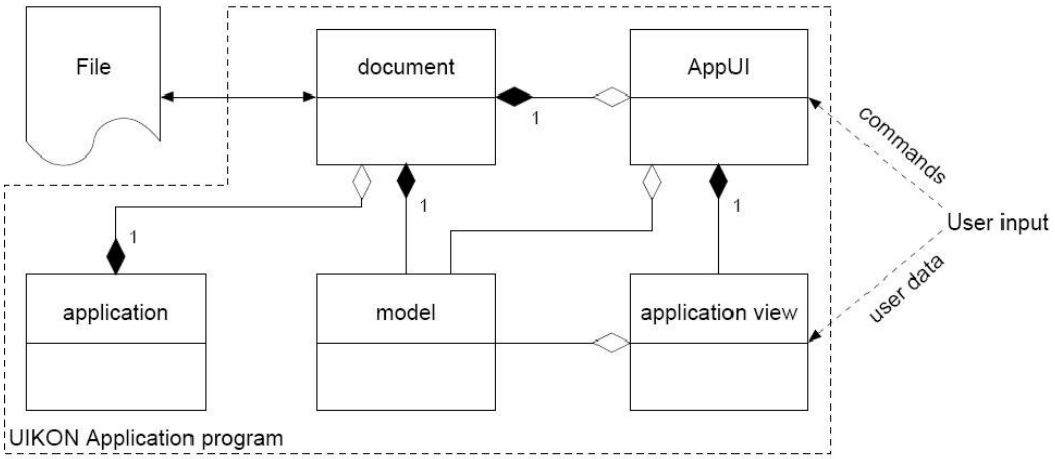
Recordemos ante todo que existen dos grandes familias de dispositivos móviles que utilizan Symbian, Series X de Nokia y UIQ, cuya gran diferencia radica en la implementación de la interfaz gráfica. El framework por defecto en Symbian para la interfaz gráfica se llama **UIKON** y provee una serie de clases y funciones. Sin embargo dicho framework ha sido derivado por **S60** y **UIQ** para añadir su funcionalidad adicional característica, dando lugar, respectivamente, a **AVKON** y **QIKON**.

Los tipos de archivo utilizados por una aplicación de interfaz gráfica son:

- Código fuente de C++ (.cpp)
- Cabeceras de C++ (.h)
- Archivos de recurso (.rss)
- Cabeceras de recurso (.hrh)

Para realizar una aplicación de interfaz gráfica hay que ceñirse a un patrón dado, creando unas clases determinadas, con una funcionalidad específica. La mayoría de los entornos de programación, como Carbide.c++, poseen plantillas

que son muy útiles a la hora de comenzar a programar una aplicación nueva ya que, directamente, genera todas las clases necesarias acordes con el patrón **UIKON** (**AVKON** o **QIKON** dependiendo del entorno en el que nos encontremos) con lo que sólo tenemos que añadir nuestra funcionalidad propia a la básica ya existente. A continuación se muestra el patrón que debe seguir una aplicación **UIKON**:



- **Application**: lanza la aplicación. CAknApplication en AVKON.

- **Document**: contiene los datos de la aplicación, está ligado a un modelo dado. CAknDocument en AVKON.

- **AppUI**: maneja los comandos transmitidos por el usuario. CAknAppui en AVKON.

- **Application View**: muestra los datos de usuario almacenados en el modelo. CCoeControl en AVKON.

Por otro lado, los archivos de recurso (**.rss**) contienen definiciones de elementos de la interfaz gráfica y, por tanto, separan su definición del código fuente en sí. Tienen una sintaxis concreta que viene definida en la documentación de Symbian.

Los archivos de cabecera de recursos (**.hrh**) permiten la utilización de los recursos definidos en los **.rss**, dentro del código fuente.

Más adelante, en los apartados dedicados al diseño y la implementación de la aplicación, se explicará cómo introducir nuestra propia funcionalidad dentro de este patrón, para poder hacer uso de la interfaz gráfica.

Compilación e instalación en un dispositivo real

Las aplicaciones en un dispositivo Symbian se instalan con la ayuda de un archivo **".sis"**, que contiene todos los archivos necesarios del programa, empaquetados, y toda la información necesaria respectiva a ellos. En este apartado vamos a ver cómo generar el archivo **".sis"**.

A la hora de instalar el emulador EPOC, se instalan también dos herramientas básicas para la generación de archivos **".sis"**: **Createsis** sirve para generar instaladores para teléfonos que usan la versión de Symbian 9 o mayor; existe también **Makesis** para dispositivos que usan una versión anterior de Symbian 9.

Se trata de dos herramientas de línea de comandos, que usaremos desde una consola de Windows. La diferencia básica es que, a partir de Symbian 9, se requiere que los programas estén firmados para poder ser instalados en el dispositivo; **Createsis** contiene opciones para el manejo de certificados y firmas, mientras que **Makesis** no necesita nada esto. Ambas herramientas deben recibir un fichero **".pkg"** con información sobre los archivos de la aplicación a incluir; más adelante se explicará la estructura de un fichero **".pkg"**.

Para poder firmar el fichero **".sis"**, es necesario obtener un certificado de Symbian del sitio www.symbiansigned.com, mediante el uso de una herramienta, **Developer Certificate Request Tool**, en la que hay introducir cierta información como, por ejemplo, el código IMEI del teléfono en el que se quiere instalar la aplicación.

Como en la realización de este proyecto no se ha dispuesto de ningún dispositivo real, no se han realizado los pasos de obtención de certificado ni de firma del archivo **".sis"**; el archivo **".sis"** proporcionado ha sido generado con la ayuda de la herramienta **Makesis** y, por tanto, no está firmado y puede ocasionar advertencias a la hora instalar en un dispositivo con Symbian 9 o mayor. Se emplaza al lector a consultar el sitio citado, para obtener una información más extensa sobre el proceso de firmado. A continuación se detalla la estructura de un archivo **".pkg"**, que servirá para la generación del archivo **".sis"**.

Dicho archivo **".pkg"** consta de unas cabeceras que contienen información sobre el idioma del programa, su versión y su creador, entre otros.

A continuación, en el cuerpo del fichero, se indican las rutas de los archivos del programa, que han sido generados dentro del directorio del emulador EPOC, y su correspondencia dentro del sistema de ficheros del teléfono móvil.

Para finalizar, se incluyen unas informaciones suplementarias, como pueden ser ficheros que contienen notas para mostrar en el momento de la instalación del programa (licencia, fichero "léame", etc.).

Vulnerabilidades Bluetooth en dispositivos móviles

En el mes de Noviembre de 2003 Adam Laurie de *A.L. Digital Ltd.* descubrió que había serios defectos en los mecanismos de autenticación y transferencia de datos en algunos dispositivos con Bluetooth habilitado. Concretamente, se encontraron tres vulnerabilidades:

- Se podían **obtener datos confidenciales** de algunos teléfonos con Bluetooth habilitado de forma anónima y sin el conocimiento ni consentimiento del propietario. Estos datos incluían, al menos, toda la agenda y el calendario, y el IMEI del teléfono.

- Posible **acceso al contenido completo de la memoria** de algunos teléfonos desde un dispositivo "de confianza", aunque este hubiese sido borrado ya de la lista de teléfonos "de confianza". Estos datos no solo incluían la agenda y el calendario, sino también archivos multimedia como imágenes y mensajes de texto.

- Se podría conseguir **acceso al conjunto de comandos AT del dispositivo**, consiguiendo llegar a los comandos y canales de más alto nivel, como datos, voz y mensajería.

Todos los dispositivos GSM y estaciones móviles UMTS o estaciones móviles multimodo deben tener un identificador unívoco, conocido originalmente como *International Mobile Equipment Identity Number* (IMEI). *British Approvals Board of Telecommunications* (BABT) fija el IMEI y mantiene un registro mostrando las asignaciones a un fabricante en concreto para usar con cada modelo determinado. BABT envía las asignaciones a la *GSM Association* para incorporarlas dentro de su *Type Accreditation Database* (TAD) / *Central Equipment Identity Register* (CEIR).

El IMEI identifica unívocamente una estación móvil individual y así crea una forma de controlar el acceso a las redes GSM basadas en distintos tipos de estaciones móviles. La estructura completa del IMEI es la siguiente:

NNXXXX XX ZZZZZZ A

Donde:

TAC - *Type Allocation Code*
NN - *Reporting Body Identifier*, por ejemplo: BABT
XXXXXX – Identificador de tipo definido por BABT. Esto limita la producción a un millón de terminales GSM por TAC. Pueden ser asignados TAC adicionales para permitir la producción de más de un millón de unidades.
ZZZZZZ - Fijado por BABT pero asignado a las estaciones móviles individuales por el fabricante.

A -

Valor 1 => 0

Valor 2 o 2+ y sistema 3GPP => Digito de control, que es definido por una función de todos los otros dígitos del IMEI

Las especificaciones de ETSI GSM requieren que toda estación móvil tenga un único IMEI. El IMEI es usado con propósitos de seguridad como parte de un control de robos o como identificador de estaciones móviles que podrían estar causando problemas técnicos en una red móvil.

Blueprinting

Blueprinting es un método para descubrir remotamente detalles acerca de dispositivos con Bluetooth habilitado. Blueprinting puede ser usado para generar estadísticas acerca de fabricantes y modelos, y para descubrir si hay o no dispositivos en un rango que tengan problemas de seguridad con Bluetooth.

Todo dispositivo que cuente con Bluetooth tiene algunas características que le hacen único, especifican el fabricante y el modelo. La dirección de un dispositivo Bluetooth es univoca y consta de 6 bytes (frecuentemente denotada como las direcciones MAC: MM:MM:MM:XX:XX:XX). Esta dirección puede ser también interpretada como la dirección hardware que es codificada en el chipset del dispositivo. Blueprinting combina la información que rebela un dispositivo Bluetooth para determinar el modelo y el fabricante. Entre otras características, es también posible hallar la versión del firmware que se implementa en un cierto dispositivo.

Bluesnarfing

Es posible, en dispositivos de ciertas marcas, conectar con dicho dispositivo sin alertar al propietario, consiguiendo acceso a partes restringidas de los datos almacenados, incluyendo la agenda en su totalidad (y cualquier imagen u otros datos que vayan asociados con las entradas de la misma), calendario, reloj, propiedades, registro de cambio, IMEI...

Algunos modelos de teléfonos móviles en los que ha sido encontrada esta vulnerabilidad son los siguientes:

- Nokia 6310i
- Nokia 8910i
- Ericsson T39
- Ericsson R520
- Ericcson T68

- Sony Ericsson T68i
- Sony Ericsson T610
- Sony Ericsson T630
- Sony Ericsson Z600

Bluesmack

BlueSmack es un ataque por el puerto Bluetooth que deja inoperativos de forma inmediata algunos dispositivos que disponen de dicho puerto. Este ataque de denegación de servicios puede ser realizado usando herramientas estándar del paquete de utilidades de BlueZ para Linux.

El "Ping de la muerte" es básicamente un mensaje de ping que afectaba a las primeras versiones de Microsoft Windows 95. El BlueSmack es el mismo tipo de ataque, pero trasladado a Bluetooth. En la capa L2CAP existe la posibilidad de solicitar un eco desde otro terminal con Bluetooth. Al igual que con el ping ICMP, la idea del ping L2CAP (solicitud de eco) es también comprobar la conectividad y tantear el tiempo de vuelta en el enlace establecido.

La utilidad *l2ping* que viene en la distribución estándar de las utilidades BlueZ permite al usuario especificar la longitud del paquete que será enviado al otro terminal (mediante la opción -s del comando de llamada a dicha utilidad).

Backdoor

El ataque por Backdoor (Puerta trasera) implica establecer una relación de confianza entre dos dispositivos, pero asegurando que esta no continuará apareciendo en el registro del dispositivo objetivo una vez finalice al ataque. De esta forma, a menos que el propietario del terminal objetivo lo este observando en el momento justo en que se establezca la conexión, este no apreciara ningún cambio, y el atacante estará capacitado para continuar usando cualquier recurso del dispositivo con el que se ha establecido la relación de confianza. Esto significa no solo la posibilidad de obtención de datos, sino de otros servicios como empleo del terminal como modem, Internet, WAP y GPRS, los cuales pueden ser conseguidos sin el consentimiento del propietario.

Además, una vez que la puerta trasera esta instalada, será posible realizar un ataque de SNARF que anteriormente nos podría haber sido denegado, y sin las restricciones de un ataque de Snarfing normal.

Blooover

Desde el experimento de BlueSnarf de Adam Laurie y el consecuente experimento de BlueBug quedó demostrado que algunos dispositivos con Bluetooth contaban con ciertos problemas de seguridad. En un primer momento, los atacantes necesitaban portátiles para el acceso a la información de terceras personas, sin embargo Blooover es una herramienta de prueba de concepto destinada a ser ejecutada en teléfonos móviles capaces de ejecutar aplicaciones J2ME. Esta puede ser ejecutada en todo teléfono que cuente con una maquina virtual J2ME MIDP 2.0 VM y una implementación de la API JSR-82 (importante para el acceso Bluetooth). Algunos de estos teléfonos son, por ejemplo, Nokia 6600, Nokia 7610, Sony Ericsson P900 y Siemens S65.

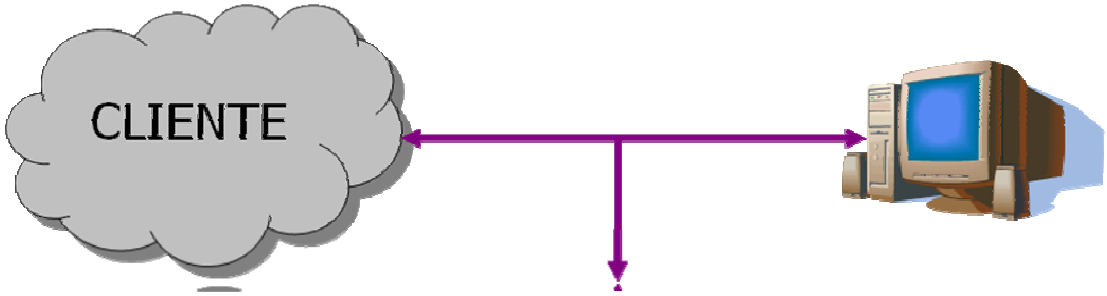
Inicialmente, Blooover trata de ser una herramienta de auditoria que la gente podría usar para comprobar si un teléfono es vulnerable. Por lo tanto, no es posible el envío de mensajes SMS y tan solo es posible iniciar y realizar llamadas a números de coste gratuito.

Bluebug

Explotando este agujero de seguridad se permite la descarga de la agenda y la lista de llamadas, el envío y lectura de mensajes SMS desde el teléfono del atacante y otras acciones sin autorización. Bajo las condiciones ideales, un ataque por BlueBug requiere solo unos pocos segundos. Dado que la vulnerabilidad que aprovecha BlueBug permite ejecutar comandos AT a través de un canal encubierto sobre un teléfono sin alertar a su propietario, este fallo de seguridad permite un gran aprovechamiento cuando el teléfono destino es atacado por Bluetooth:

- Inicio de llamadas telefónicas
- Envío de mensajes SMS a cualquier numero
- Lectura de mensajes SMS del teléfono victima
- Lectura de las entradas de la agenda
- Escritura de entradas en la agenda
- Configuración de reenvío de llamadas
- Conexión a Internet
- Forzar al teléfono a usar una determinada compañía telefónica

Características de un Sniffer



¿Qué es un sniffer?

De igual manera que los circuitos de teléfono, las redes de ordenadores son canales de comunicaciones compartidos. Es simplemente demasiado costoso poner un *switch* (*hub*) para cada par de ordenadores implicados en la comunicación. El compartir significa que las computadoras pueden recibir la información que fue enviada a otras máquinas. Al capturar la información que pasa la red se llama el *sniffing*.

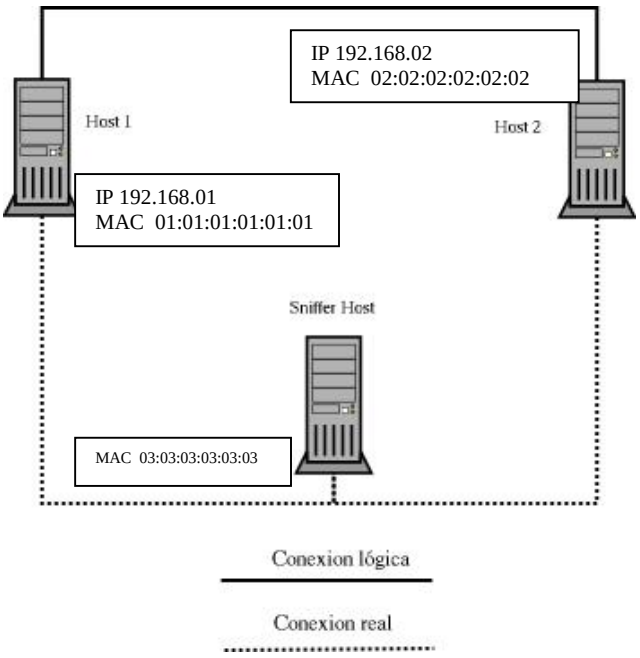
Normalmente la manera de conectar varios ordenadores es mediante Ethernet. El protocolo de Ethernet trabaja enviando la información del paquete a todos los *hosts* en el mismo circuito. La cabecera del paquete contiene la dirección apropiada de la máquina destino. Solamente la máquina con la dirección que va en la cabecera se supone que va a aceptar el paquete. Una máquina que está aceptando todos los paquetes, sin importar lo que ponga en la cabecera del paquete, se dice que está en modo promiscuo.

Debido a que en un ambiente normal del establecimiento de una red, la cuenta y la información de la contraseña se pasa a lo largo de Ethernet en *texto claro*, no es complicado para un atacante una vez que obtengan el *root*, poner una máquina en modo promiscuo (*sniffing*). Esto compromete todas las máquinas en la red.

Otra forma de monitorear el tráfico de una red ethernet es utilizar la técnica llamada "*arp-spoofing*". Este método no pone la interfaz de red en modo

promiscuo. Esto no es necesario porque los paquetes son para nosotros y el switch enrutará los paquetes hacia nosotros.

El método consiste en “envenenar” la cache arp de las dos máquinas que queremos monitorear. Una vez que las caches estén envenenadas, los dos hosts comenzarán la comunicación, pero los paquetes serán para nosotros, los monitorearemos y los enrutaremos de nuevo al host apropiado. De esta forma la comunicación es transparente para los dos hosts. El esquema de la comunicación es sencillo:



Desde nuestra máquina enviaremos paquetes de tipo arp-reply falsos a las dos host que queremos monitorear. En estos reply’s debemos de decirle al host 1 que la dirección ethernet del segundo host es la nuestra, quedando esta información almacenada en su cache arp. Este equipo enviará ahora los paquetes al host 2 pero con nuestra dirección MAC. Los paquetes ya son nuestros. El switch se encargará de hacernos llegar los datos.

Como ejemplo en el esquema anterior enviamos un flujo constante de arp-reply (para evitar que la cache arp de las maquinas se refresque con la información verdadera) al host 1 y host 2 con los siguientes datos:

HOST 1: arp-reply informando que 192.168.0.2 tiene dirección MAC 03:03:03:03:03:03

HOST 2 : arp-reply informando que 192.168.0.1 tiene dirección MAC 03:03:03:03:03:03

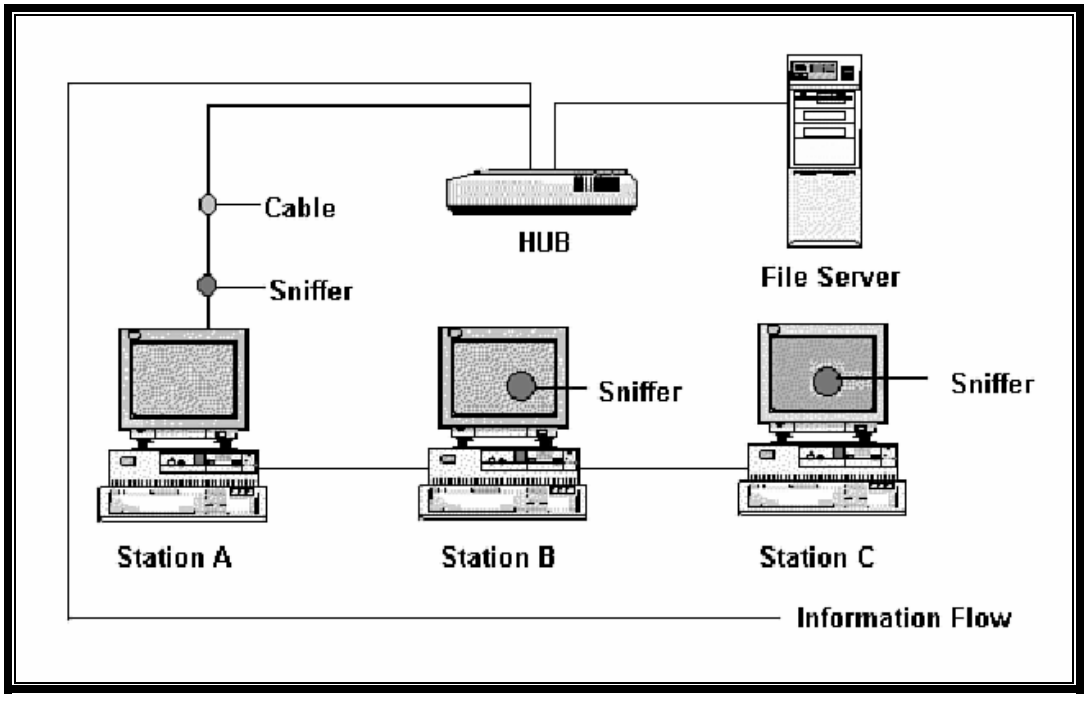
De esta forma estamos “envenenando” las cache arp. A partir de ahora lo paquetes que se envíen entre ambas nos llegarán a nosotros, pero para que ambos hosts no noten nada extraño, deberemos de hacer llegar los paquetes a su destino final. Para ello deberemos de tratar los paquetes que recibamos en función del host de origen:

Paquetes procedentes de HOST 1 ----------> reenviar a 02:02:02:02:02:02

Paquetes procedentes de HOST 2 ----------> reenviar a 01:01:01:01:01:01

De esta forma la comunicación entre ambos no se ve interrumpida, y podemos observar todo el tráfico entre ellos. Solo tendremos que utilizar un sniffer para poder capturar y filtrar el tráfico entre ambos.

Las comunicaciones entre ordenadores consisten en datos binarios aparentemente al azar. Por lo tanto, los sniffers vienen con una característica conocida como el "análisis del protocolo", que les permite "descifrar" el tráfico del ordenador y hacer que tengan sentido todos esos datos binarios capturados de los paquetes.



¿Para que se usa un sniffer?

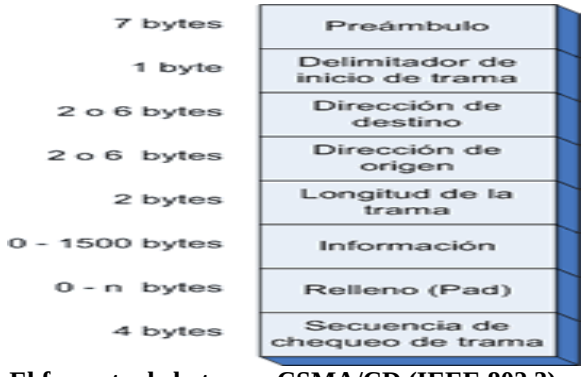
Los programas de sniffers han estado ejecutándose por la red durante mucho tiempo en dos formas. Los programas comerciales de sniffers se usan a menudo para ayudar en el mantenimiento de las redes. Mientras que sniffers "underground" son usados por los crackers para introducirse en los ordenadores ajenos. Algunos usos típicos de estos programas son:

- Captura de passwords y logins que están en texto plano (sin encriptar) desde la red.
- Conversión de datos a un formato comprensible por los humanos
- Análisis de errores para descubrir problemas en la red.
- Análisis de rendimiento para descubrir posibles cuellos de botella en la red.
- Detección de intrusos en la red para hackers/crackers potenciales

Un sniffer más que una herramienta de ataque en manos de un administrador de red puede ser una valiosa arma para la auditoria de seguridad en la red. Puesto que el acceso a la red externa debe estar limitado a un único punto. Un sniffer puede ser la herramienta ideal para verificar como se está comportando la red.

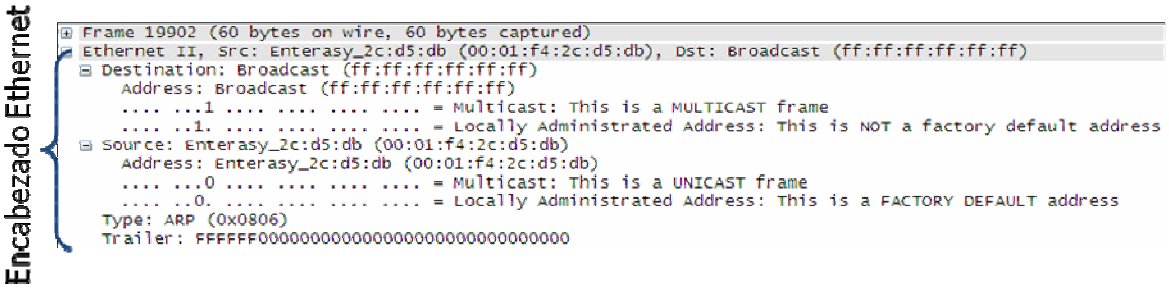
Ethereal:

Ethereal es un sniffer bastante conocido y gratuito que podemos obtener de su página web http://www.ethereal.com/. A continuación vamos a definir varios términos y donde los encontramos en las diferentes pantallas de este sniffer.

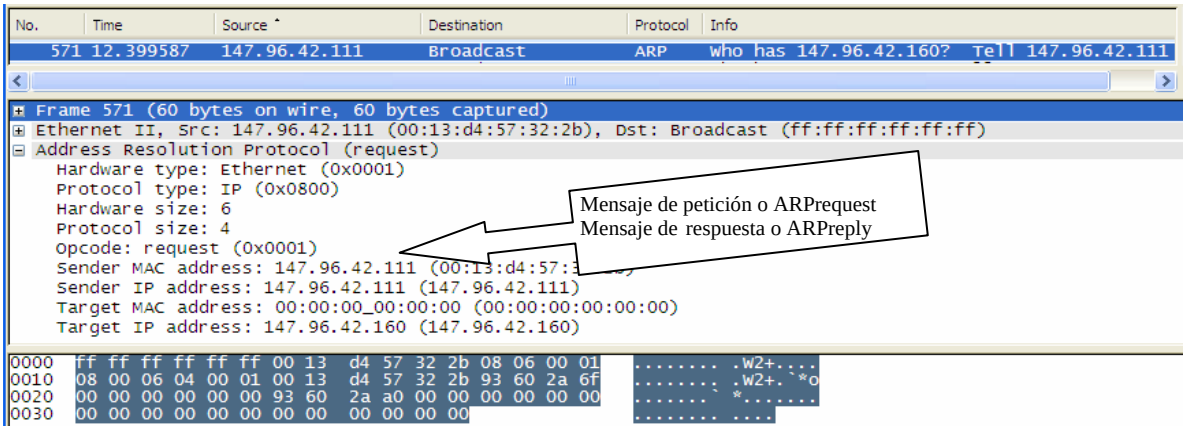


El formato de la trama CSMA/CD (IEEE 802.3)

Encabezado Ethernet en una de las pantallas de Ethereal.



La trama **ARP** (Address Resolution Protocol). El protocolo de resolución de direcciones es responsable de convertir las dirección de protocolo de alto nivel(direcciones IP) a direcciones de red físicas.



ARP en Etheral

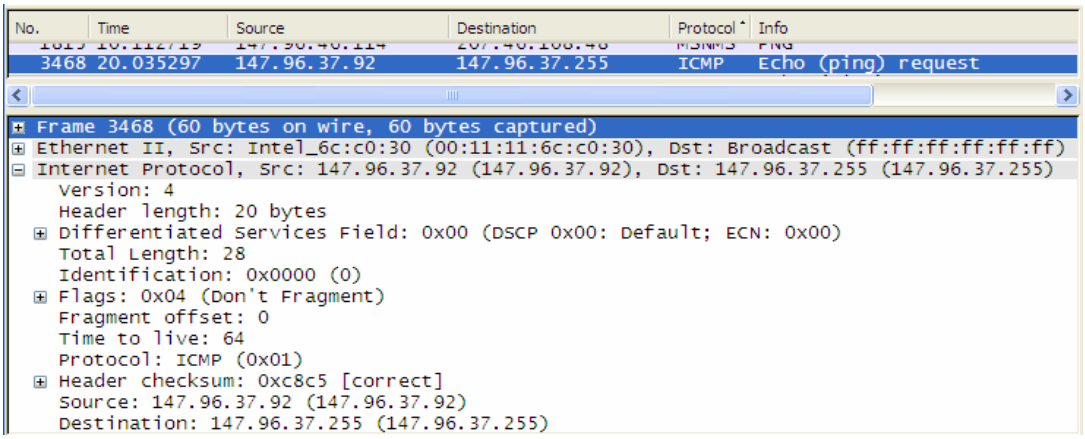
IP (Internet Protocol).

Porta datagramas de la fuente al destino. El nivel de transporte parte el flujo de datos en datagramas. Durante su transmisión se puede partir un datagrama en fragmentos que se montan de nuevo en el destino.

Proporciona un servicio de distribución de paquetes de información orientado a no conexión de manera no fiable. La orientación a no conexión significa que los paquetes de información, que será emitido a la red, son tratados

independientemente, pudiendo viajar por diferentes trayectorias para llegar a su destino. El término no fiable significa más que nada que no se garantiza la recepción del paquete.
La unidad de información intercambiada por IP es denominada datagrama. Tomando como analogía los marcos intercambiados por una red física los datagramas contienen un encabezado y un área de datos. IP no especifica el contenido del área de datos, ésta será utilizada arbitrariamente por el protocolo de transporte.

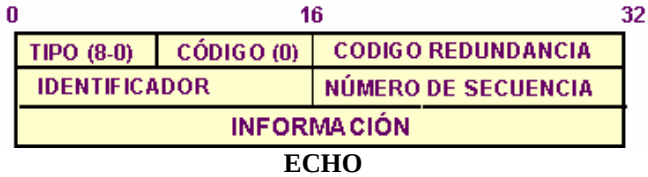
Formato de la trama IP



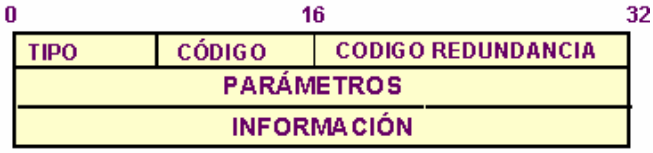
Trama IP en Ethereal

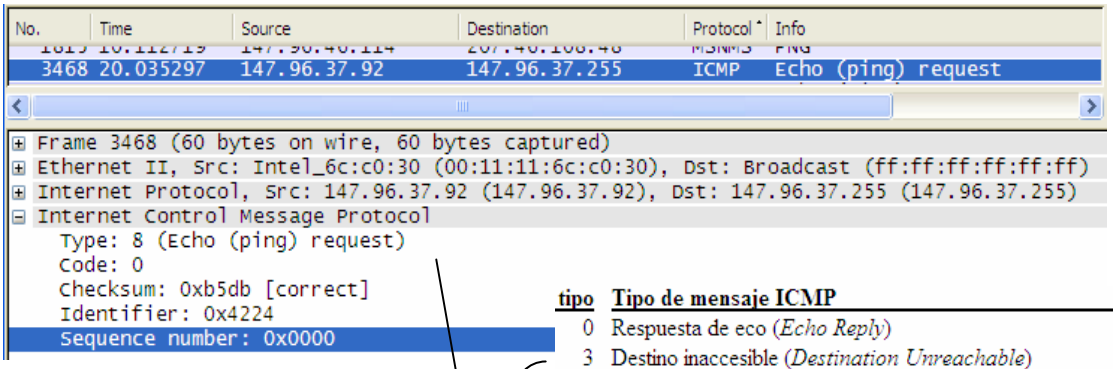
ICMP (Protocolo de Mensajes de Control y Error de Internet)

El protocolo ICMP solamente informa de incidencias en la entrega de paquetes o de errores en la red en general, pero no toma decisión alguna al respecto. Esto es tarea de las capas superiores.



ECHO

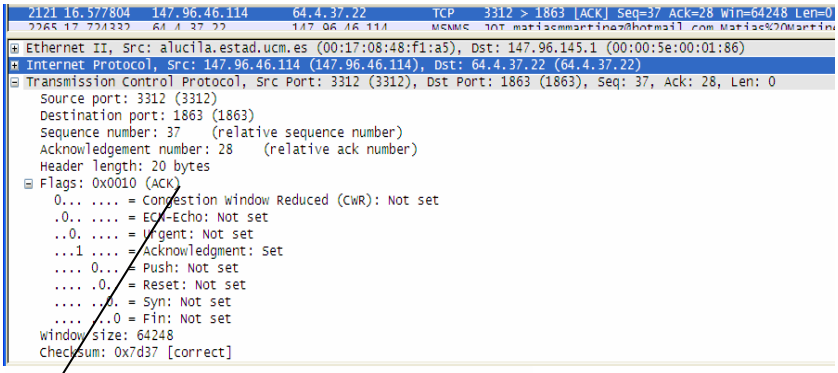




tipo	Tipo de mensaje ICMP
0	Respuesta de eco (*Echo Reply*)
3	Destino inaccesible (*Destination Unreachable*)
4	Disminución del tráfico desde el origen (*Source Quench*)
5	Redireccionar (cambio de ruta) (*Redirect*)
8	Solicitud de eco (*Echo*)
11	Tiempo excedido para un datagrama (*Time Exceeded*)
12	Problema de Parámetros (*Parameter Problem*)
13	Solicitud de marca de tiempo (*Timestamp*)
14	Respuesta de marca de tiempo (*Timestamp Reply*)
15	Solicitud de información (obsoleto) (*Information Request*)
16	Respuesta de información (obsoleto) (*Information Reply*)
17	Solicitud de máscara (*Addressmask*)
18	Respuesta de máscara (*Addressmask Reply*)

TCP (Transmision Control Protocol)

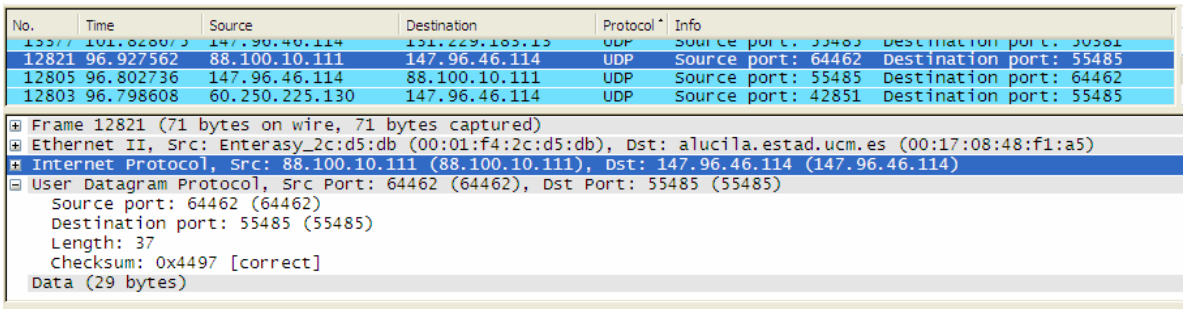
Trama TCP



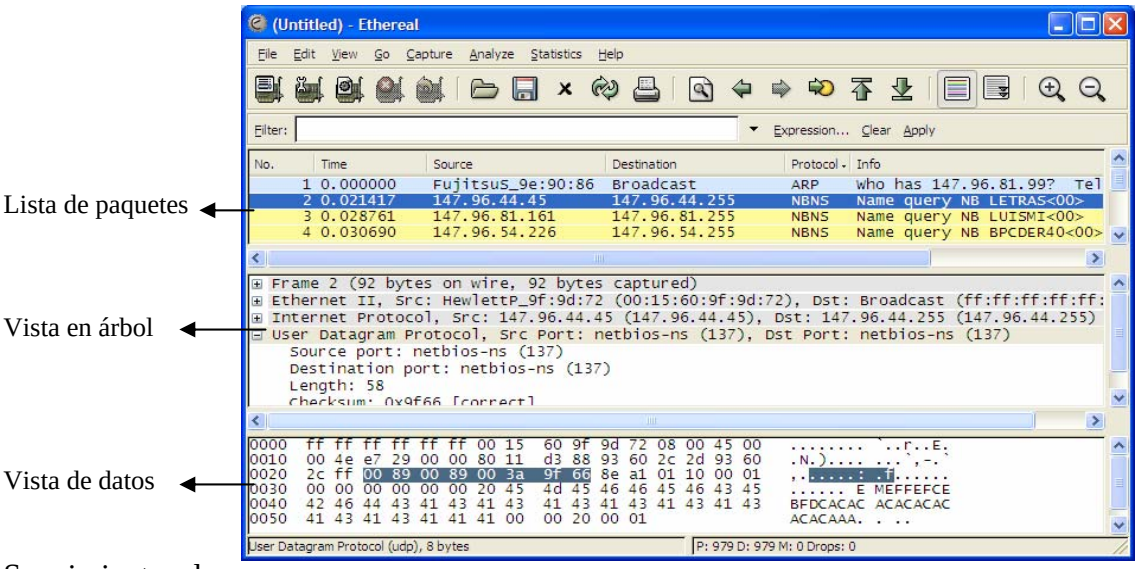
URG. *Puntero de urgencia*
ACK. *Número de acuse de recibo.*
PSH. La aplicación ha solicitado una operación *push(Priorizar)*
RST. Interrupción de la conexión actual.
SYN. Sincronización de los números de secuencia.
FIN. Indica al otro extremo que la aplicación ya no tiene más datos para enviar.

UDP (User Datagram Protocol)

0	16	32
PUERTO ORIGEN	PUERTO DESTINO	
LONGITUD MENSAJE UDP	CODIGO REDUNDANCIA	
DATOS		



Ventana principal de Etheral

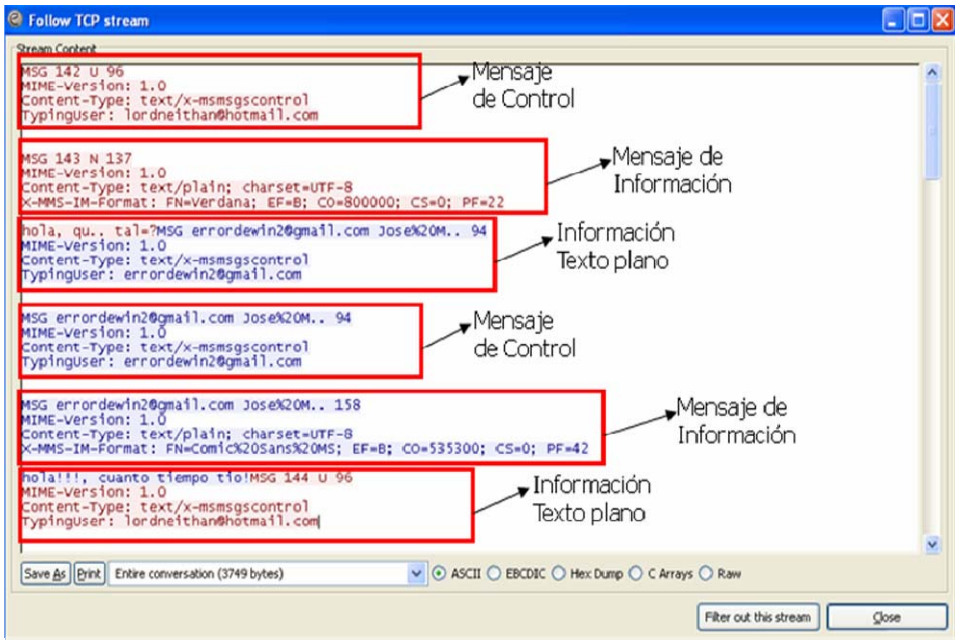


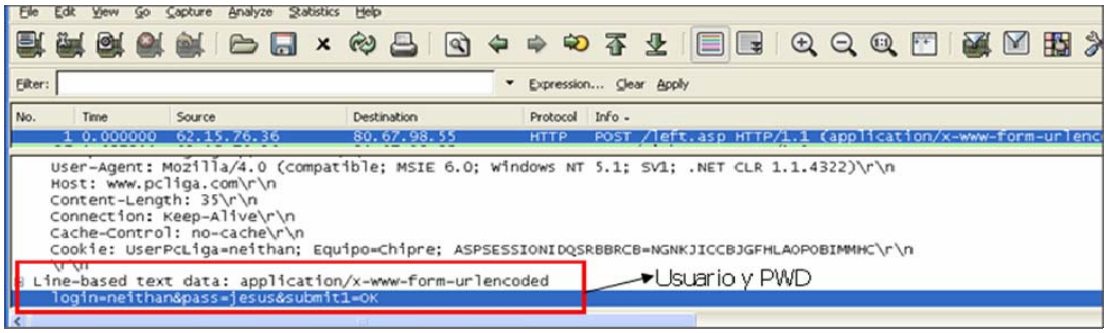
Lista de paquetes

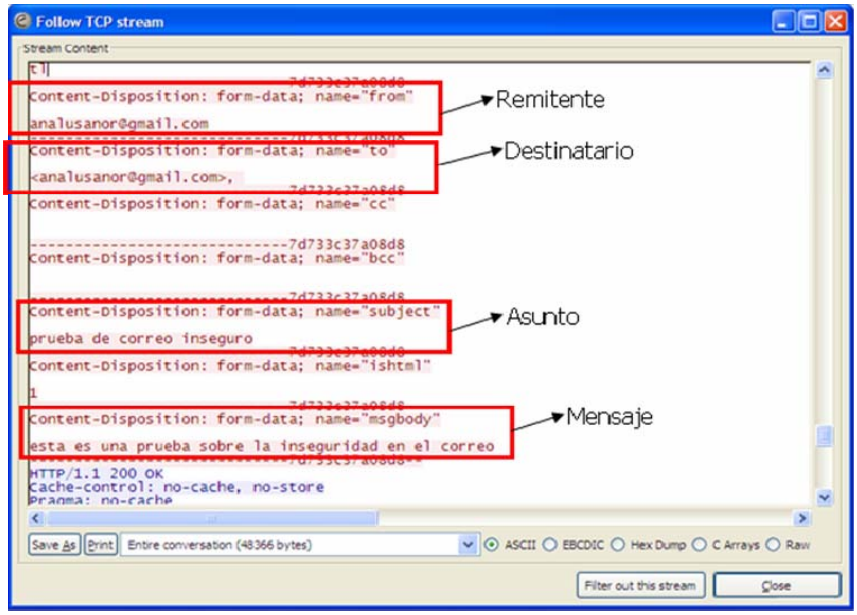
Vista en árbol

Vista de datos

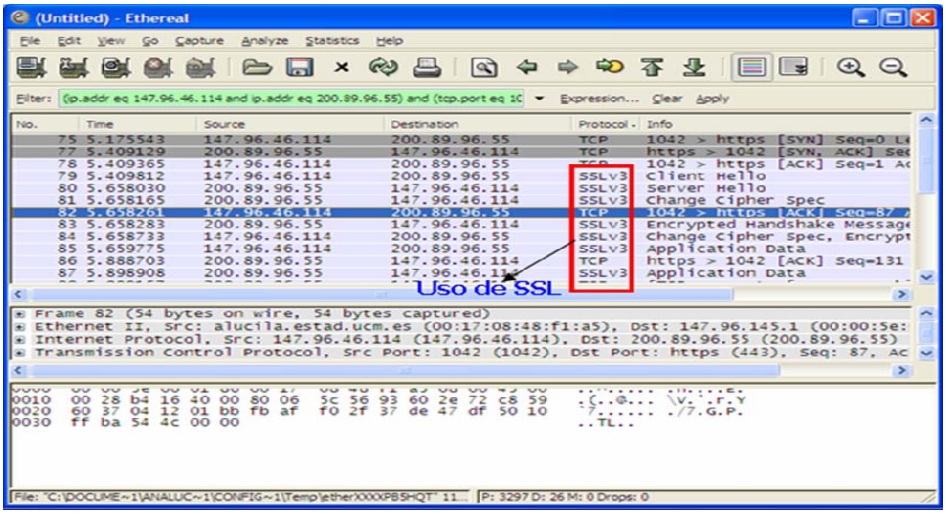
Seguimiento de trazas TCP


Follow TCP stream
Stream Content
MSG 142 U 96
MIME-Version: 1.0
Content-Type: text/x-msmsgscontrol
TypingUser: lordneithan@hotmail.com
Mensaje de Control
MSG 143 N 137
MIME-Version: 1.0
Content-Type: text/plain; charset=UTF-8
X-MMS-IM-Format: FN=Verdana; EF=B; CO=800000; CS=0; PF=22
Mensaje de Información
hola, qu.. tal=?MSG errordewin2@gmail.com Jose%20M.. 94
MIME-Version: 1.0
Content-Type: text/x-msmsgscontrol
TypingUser: errordewin2@gmail.com
Información Texto plano
MSG errordewin2@gmail.com Jose%20M.. 94
MIME-Version: 1.0
Content-Type: text/x-msmsgscontrol
TypingUser: errordewin2@gmail.com
Mensaje de Control
MSG errordewin2@gmail.com Jose%20M.. 158
MIME-Version: 1.0
Content-Type: text/plain; charset=UTF-8
X-MMS-IM-Format: FN=Comic%20Sans%20MS; EF=B; CO=535300; CS=0; PF=42
Mensaje de Información
hola!!!, cuanto tiempo tio!MSG 144 U 96
MIME-Version: 1.0
Content-Type: text/x-msmsgscontrol
TypingUser: lordneithan@hotmail.com
Información Texto plano
Save As
Print
Entire conversation (3749 bytes)
ASCII
EBCDIC
Hex Dump
C Arrays
Raw
Filter out this stream
Close


File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
No. Time Source Destination Protocol Info
1 0.000000 62.15.76.36 80.67.98.55 HTTP POST /left.asp HTTP/1.1 (application/x-www-form-urlenc
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)\r\n
Host: www.pcliga.com\r\n
Content-Length: 35\r\n
Connection: Keep-Alive\r\n
Cache-Control: no-cache\r\n
Cookie: UserPcLiga=neithan; Equipo=Chipre; ASPSESSIONIDQSRBBRCB=NGNKJICCBJGFHLAOPOBIMMHC\r\n
\r\n
Line-based text data: application/x-www-form-urlencoded
login=neithan&pass=jesus&submit1=OK
Usuario y PWD


Follow TCP stream
Stream Content
Content-Disposition: form-data; name="from"
analusanor@gmail.com
Remitente
Content-Disposition: form-data; name="to"
<analusanor@gmail.com>,
Destinatario
Content-Disposition: form-data; name="cc"
Content-Disposition: form-data; name="bcc"
Content-Disposition: form-data; name="subject"
prueba de correo inseguro
Asunto
Content-Disposition: form-data; name="ishtml"
1
Content-Disposition: form-data; name="msgbody"
esta es una prueba sobre la inseguridad en el correo
Mensaje
HTTP/1.1 200 OK
Cache-control: no-cache, no-store
Pragma: no-cache
Save As
Print
Entire conversation (48366 bytes)
ASCII
EBCDIC
Hex Dump
C Arrays
Raw
Filter out this stream
Close

En esta captura podemos observar que se esta usando SSL.



Módulos de un sniffer.

1. El hardware

La mayoría de los productos trabajan con las tarjetas de red estándar, aunque algunos requieren un hardware especial. Si se usa algún tipo de hardware especial, se puede analizar fallos como errores CRC, problemas de voltaje, programas de cable, etc.

2. Driver de captura

Ésta es la parte más importante. Captura el tráfico de la red desde el cable, lo filtra según se desee y luego almacena los datos en el buffer.

3. Buffer

Una vez que los paquetes son capturados desde la red, se almacenan en un buffer. Hay dos modos de captura distintos: captura hasta que el buffer se llene o usar el buffer como un “round robin” donde los datos más recientes reemplazan a los más antiguos.

4.Decodificar

Esta opción muestra el contenido del tráfico de la red con un texto descriptivo para que el analista sepa qué está pasando.

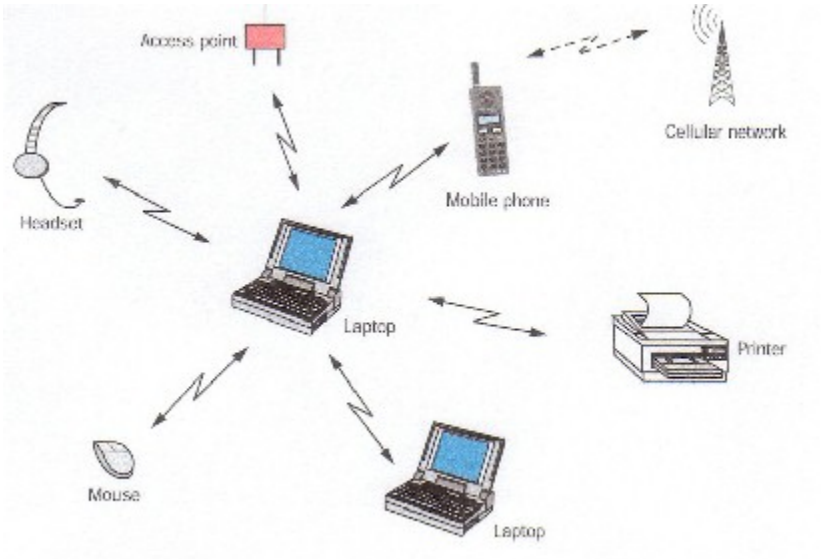
Introducción a Bluetooth

¿Qué es Bluetooth?

Es la norma que define un estándar global de comunicación inalámbrica, que posibilita la transmisión de voz y de datos entre diferentes equipos mediante un enlace por radiofrecuencia. Los principales objetivos que se pretende conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

La tecnología Bluetooth comprende hardware, software y requerimientos de interoperabilidad, por lo que para su desarrollo ha sido necesaria la participación de los principales fabricantes de los sectores de las telecomunicaciones y la informática, tales como: **Ericsson**, **Nokia**, **Toshiba**, **IBM**, **Intel** y otros. Posteriormente se han ido incorporando muchas más compañías, y últimamente también se han incorporado empresas de sectores tan variados como: automatización industrial, maquinaria, ocio y entretenimiento, fabricantes de juguetes, electrodomésticos, etc., con lo que en poco tiempo se nos presentará un panorama de total conectividad de nuestros aparatos tanto en casa como en el trabajo.



Antecedentes

En 1994 Ericsson inició un estudio para investigar la viabilidad de una interface vía radio, de bajo coste y bajo consumo, para la interconexión entre teléfonos móviles y otros accesorios con la intención de eliminar cables entre aparatos. El estudio partía de un largo proyecto que investigaba sobre unos multi-comunicadores conectados a una red celular. Se llegó a un enlace de radio de corto alcance llamado *MC link.* Conforme este proyecto avanzaba se fue viendo claro que este tipo de enlace podía ser utilizado ampliamente en un gran número de aplicaciones, ya que tenía como principal virtud el que se basaba en un chip de radio relativamente económico.

El SIG

A comienzos de 1997, según avanzaba el proyecto *MC link*, Ericsson fue despertando el interés de otros fabricantes de equipos portátiles. Rápidamente se vio claramente que para que el sistema tuviera éxito, un gran número de equipos deberían estar equipados con esta tecnología. Esto fue lo que originó, a principios de 1998, la creación de un grupo de interés especial (**SIG**), formado por 5 promotores: Ericsson, Nokia, IBM, Toshiba e Intel. La idea era lograr un conjunto adecuado de áreas de negocio: dos líderes del mercado de las telecomunicaciones, dos líderes del mercado de los PCS portátiles y un líder de la fabricación de chips. El propósito principal del consorcio fue y es el establecer un estándar para la *interface* aérea junto con su software de control, con el fin de asegurar la interoperabilidad de los equipos entre los diversos fabricantes.

La interface aérea Bluetooth

El primer objetivo para los productos Bluetooth de primera generación eran los entornos de la gente de negocios que viaja frecuentemente. Por lo que se debería pensar en integrar el chip de radio Bluetooth en equipos como: PCS portátiles, teléfonos móviles, PDAs y auriculares. Esto originaba una serie de cuestiones previas que deberían solucionarse tales como:

- El sistema debería operar en todo el mundo.
- El emisor de radio deberá consumir poca energía, ya que debe integrarse en equipos alimentados por baterías.
- La conexión deberá soportar voz y datos y, por lo tanto, aplicaciones multimedia.

Banda de frecuencia libre

Para poder operar en todo el mundo es necesaria una banda de frecuencia abierta a cualquier sistema de radio independientemente del lugar del planeta donde nos encontremos. Sólo la banda ISM (médico-científica internacional) de 2,45 GHz cumple con este requisito, con rangos que van de los 2.400 MHz a los

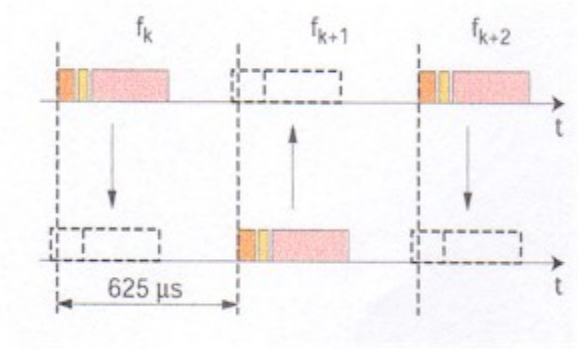
2.500 MHz, y sólo con algunas restricciones en países como Francia, España y Japón.

Salto de frecuencia

Debido a que la banda ISM está abierta a cualquiera, el sistema de radio Bluetooth deberá estar preparado para evitar las múltiples interferencias que se pudieran producir. Éstas pueden ser evitadas utilizando un sistema que busque una parte no utilizada del espectro o un sistema de salto de frecuencia. En los sistemas de radio Bluetooth se suele utilizar el método de salto de frecuencia debido a que esta tecnología puede ser integrada en equipos de baja potencia y bajo coste. Este sistema divide la banda de frecuencia en varios canales de salto, donde los transceptores durante la conexión van cambiando de uno a otro canal de salto de manera pseudoaleatoria. Con esto se consigue que el ancho de banda instantáneo sea muy pequeño y también una propagación efectiva sobre el total de ancho de banda. En conclusión, con el sistema FH (Salto de Frecuencia) se pueden conseguir transceptores de banda estrecha con una gran inmunidad a las interferencias.

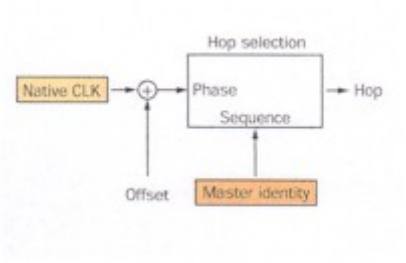
Definición de canal

Como hemos comentado, Bluetooth utiliza un sistema FH/TDD (salto de frecuencia/división de tiempo dúplex), en el que el canal queda dividido en intervalos de 625 µs, llamados slots, donde cada salto de frecuencia es ocupado por un slot. Esto da lugar a una frecuencia de salto de 1600 veces por segundo, en la que un paquete puede ocupar un slot para la emisión y otro para la recepción y que pueden ser usados alternativamente, dando lugar a un esquema de tipo TDD.



Dos o más unidades Bluetooth pueden compartir el mismo canal dentro de un **piconet.** Una unidad actúa como maestra, controlando el tráfico de datos en el piconet que se genera entre las demás unidades y las otras actúan como esclavas, enviando y recibiendo señales hacia el maestro. El salto de frecuencia del canal está determinado por la secuencia de la señal, es decir, el orden en que llegan los saltos y por la fase de esta secuencia. En Bluetooth la secuencia queda fijada por la identidad de la unidad maestra del piconet (un código único para cada equipo) y por su frecuencia de reloj. Por lo que, para que una unidad esclava pueda

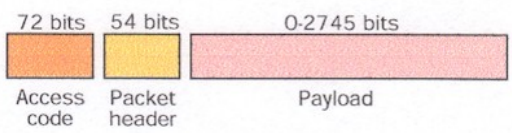
sincronizarse con una unidad maestra, ésta debe añadir un ajuste a su propio reloj nativo y así poder compartir la misma portadora de salto.



En países donde la banda está abierta a 80 canales o más, espaciados todos ellos a 1 MHz, se han definido 79 saltos de portadora, y en aquellos donde la banda es más estrecha se han definido 23 saltos.

Definición de paquete

La información que se intercambia entre dos unidades Bluetooth se realiza mediante un conjunto de slots que forman un paquete de datos. Cada paquete comienza con un código de acceso de 72 bits, que se deriva de la identidad maestra, seguido de un paquete de datos de cabecera de 54 bits. Éste contiene importante información de control, como tres bits de acceso de dirección, tipo de paquete, bits de control de flujo, bits para la retransmisión automática de la pregunta y chequeo de errores de campos de cabecera. Finalmente, el paquete que contiene la información, que puede seguir al de cabecera, tiene una longitud de 0 a 2745 bits. En cualquier caso, cada paquete que se intercambia en el canal está precedido por el código de acceso.



Los receptores del piconet comparan las señales que reciben con el código de acceso. Si éstas no coinciden, el paquete recibido no es considerado como válido en el canal y el resto de su contenido es ignorado.

Definición de enlace físico

En la especificación Bluetooth se han definido dos tipos de enlace que permitan soportar incluso aplicaciones multimedia:

- Enlace de sincronización de conexión orientada (SCO).
- Enlace asíncrono de baja conexión (ACL).

Los enlaces SCO soportan conexiones asimétricas, punto a punto, usadas normalmente en conexiones de voz. Estos enlaces están definidos en el canal, reservándose dos slots consecutivos (envío y retorno) en intervalos fijos. Los enlaces ACL soportan conmutaciones punto a punto simétricas o asimétricas, típicamente usadas en la transmisión de datos.

Un conjunto de paquetes se han definido para cada tipo de enlace físico:

- Para los enlaces SCO, existen tres tipos de slot simple, cada uno con una portadora a una velocidad de 64 kbit/s. La transmisión de voz se realiza sin ningún mecanismo de protección, pero si el intervalo de las señales en el enlace SCO disminuye, se puede seleccionar una velocidad de corrección de envío de 1/3 o 2/3.

- Para los enlaces ACL se han definido el slot-1, slot-3, slot-5. Cualquiera de los datos pueden ser enviados protegidos o sin proteger con una velocidad de corrección de 2/3. La máxima velocidad de envío es de 721 Kbit/s en una dirección y 57.6 Kbit/s en la otra.

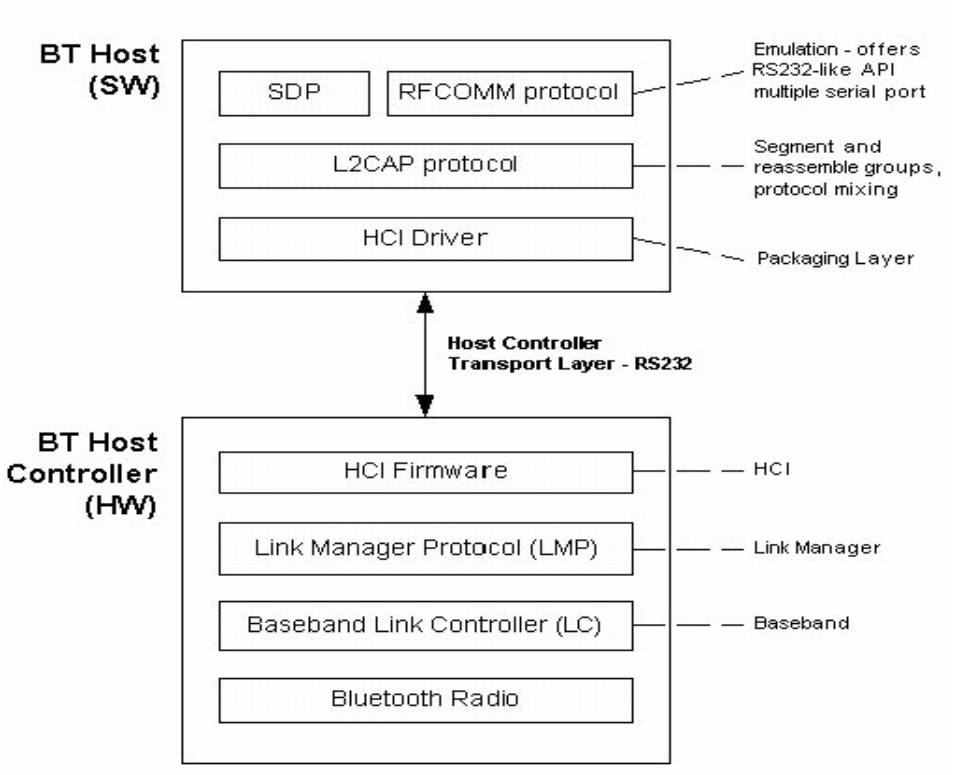
Inmunidad a las interferencias

Como se mencionó anteriormente Bluetooth opera en una banda de frecuencia que está sujeta a considerables interferencias, por lo que el sistema ha sido optimizado para evitar estas interferencias. En este caso la técnica de salto de frecuencia es aplicada a una alta velocidad y una corta longitud de los paquetes (1600 saltos/segundo para slots simples). Los paquetes de datos están protegido por un esquema ARQ (repetición automática de consulta), en el cual los paquetes perdidos son automáticamente retransmitidos. Aún así, con este sistema, si un paquete de datos no llegase a su destino, sólo una pequeña parte de la información se perdería. La voz no se retransmite nunca, sin embargo, se utiliza un esquema de codificación muy robusto. Este esquema está basado en una modulación variable de declive delta (CSVD), que sigue la forma de la onda de audio y es muy resistente a los errores de bits. Estos errores son percibidos como ruido de fondo, que se intensifica si los errores aumentan.

Bluetooth en Symbian

Revisión de la arquitectura bluetooth en Symbian.

Como en otras tecnologías de la comunicación, Bluetooth está compuesta por una jerarquía de componentes que se muestran en la siguiente figura.



Arquitectura bluetooth.

El API bluettoth para el SO Symbian proporciona acceso a RFCOMM, L2CAP, SDP, OBEX y a HCI.

Los componentes del Bluetooth Host Controller proporcionan el nivel mas bajo de la pila. Están normalmente implementados en hardware y las aplicaciones no tienen acceso directo a esta capa. Estos componentes permiten a las aplicaciones enviar o recibir datos sobre un enlace Bluetooth o configurar este enlace:

- RFCOMM permite a una aplicación tratar el enlace bluetooth de un modo similar a como si fuera sobre un puerto serie. Se usa para soportar protocolos heredados.
- Logical Link Control and Adaptation Protocol (L2CAP) permite mayor detalle de control del enlace.

- Service Discovery Protocol (SDP) se usa para localizar y descubrir servicios proporcionados o disponibles por el dispositivo bluetooth. Las aplicaciones normalmente lo usan cuando están estableciendo comunicaciones con otro dispositivo bluetooth.
- Host Controller Interface (HCI) es el que dirige de los componentes de alto nivel para comunicarse con el hardware. Los comandos de HCI proporcionan una interfaz de comandos para el controlador de la banda base y el link manager.

Módulos del API de bluetooth.

El API bluetooth puede usarse desde los siguientes módulos:

- Bluetooth Sockets: Encapsulan el acceso a L2CAP y RFCOMM a través de TCP/IP como interfaz de socket.
- Bluetooth Service Discovery Database:

Encapsula un lado del SDP. Un servicio local lo usa para registrar sus atributos, de modo que el dispositivo remoto puede descubrir su presencia y determinar si puede ser usado.

- Bluetooth Service Discovery Agent:

Encapsula el otro ladeo del SDP. Este permite al usuario descubrir que servicios están disponibles en un dispositivo remoto, así como los atributos de estos servicios.

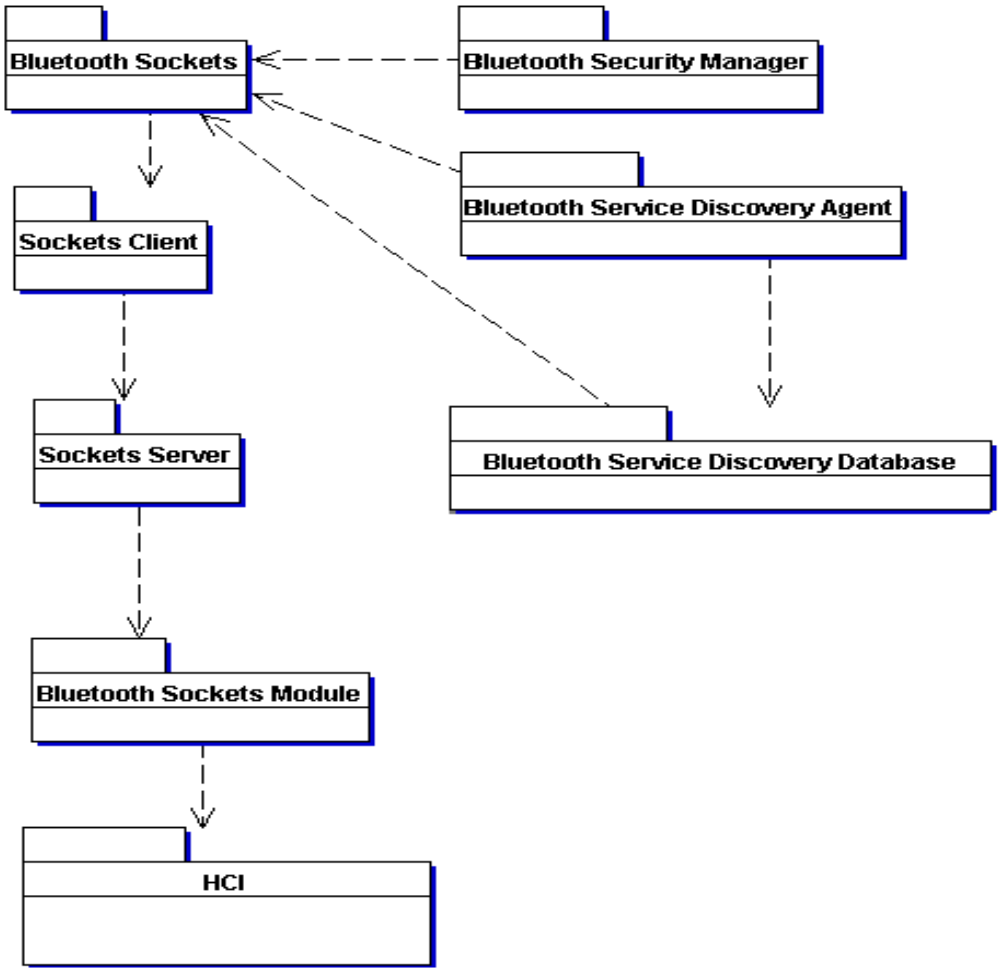
- Bluetooth Security Manager:

Habilita a los servicios con requerimientos apropiados de seguridad, con las conexiones de entrada que deben conocer.

- Bluetooth UI (también llamado Bluetooth Device Selection UI)

Bluetooth UI se usa en la selección de un dispositivo remoto. Cuando hay varios dispositivos remotos disponibles, este avisa al usuario que seleccione cual de las conexiones deberá realizar. Para hacer esto se suministra un dialogo. Los clientes solicitan un hilo en background llamado Notifier Server para crear el dialogo, el servidor puede poner un dialogo sobre el UI o cualquier aplicación que esté en primer plano.

La siguiente imagen muestra como los diferentes APIs de Bluetooth estan relacionados. Notar que el Bluetooth Sockets API es el fundamental API desde el cual los otros APIs cuentan para efectuar comunicaciones con otros dispositivos.



Architectural relationships between Symbian Bluetooth API modules.

Además de los APIs generales, existen un par de APIs especificos para la serie 60 (S60). Bluetooth Notifier API (fue introducido en S60 2nd Edición, FP1) y Send UI API, que ofrece un API de alto nivel para enviar diferentes tipos de mensajes incluyendo mensajes bluetooth. Esto se tratará posteriormente en este documento.

Bluetooth sockets

En Symbian, los sockets bluetooth se usan para descubrir otros dispositivos bluetooth y para leer y escribir datos sobre bluetooth. EL API de los sockets bluetooth soportan comunicaciones sobre las capas L2CAP y RFCOMM del protocolo bluetooh. El API esta basado en los Sockets Client Side API, que proporcionan un API estándar permitiendo a un cliente realizar una conexión con un dispositivo remoto. De manera alternativa, permite a un dispositivo actuar como un servidor y tener un dispositivo remoto conectado a él. Una vez

conectado, los dispositivos pueden enviar y recibir datos antes de que se desconecten. Este API (Bluetooth Sockets API) incluye los tipos y constantes apropiadas que permitan al API ser usado con Bluetooth.

El API tiene cinco conceptos clave: dirección del socket, petición del dispositivo remoto, comandos y opciones RFCOMM, comandos L2CAP, y comandos HCI.
Un dispositivo Bluetooth debe localizar un dispositivo adecuado para conectarse a él. El SDP (Service Discovery Protocol) efectúa estas tareas.

El SDP se puede descomponer en dos partes principales:

1. El descubrimiento de dispositivos y servicios dentro del área local.
2. El anuncio de servicios del dispositivo local.

Bluetooth Service Discovery Database

Bluetooth Service Discovery Database permite a un servicio local introducir sus propiedades dentro de un servicio local bluetooth database. Haciendo esto habilitamos a los dispositivos remotos Bluetooth descubrir que el servicio esta disponible.

Service Discovery Database API es uno de los dos APIs que permiten al desarrollador usar el Bluetooth Service Discovery Protocol. El otro, el Bluetooth Sevice Discovery Agent, capacita al desarrollador para descubrir los servicios Bluetooth y los atributos de aquellos servicios que estén disponibles en un dispositivo remoto.

Bluetooth Service Discovery Agent

Bluetooth Sevice Discovery Agent, capacita al desarrollador para descubrir los servicios Bluetooth y los atributos de aquellos servicios que estén disponibles en un dispositivo remoto.

El API de Service Discovery Agent es uno de los dos APIs que permiten al desarrollador usar el Bluetooth Service Discovery Protocol. EL otro, el Bluetooh Service Discovery Database, permite a un servicio local introducir sus propias propiedades en un servicio de base de datos local.

Bluetooth Security Manager

Permite a los servicios Bluetooth configurar los requisitos de seguridad con las conexiones que el servicio deberá realizar. La configuración de seguridad define si la autenticación, autorización y la encriptación se requieren o no. Si el dispositivo remoto requiere autenticación o encriptación para la conexión, la pila deberá tratar esto de forma transparente. El caso normal es que el dispositivo receptor (esclavo) ponga la seguridad.

Bluetooth UI

Bluetooth UI se usa en la selección de un dispositivo remoto. Cuando hay varios dispositivos remotos disponibles, este avisa al usuario que seleccione cual de las conexiones deberá realizar. Para hacer esto se suministra un dialogo. Los clientes solicitan un hilo en background llamado Notifier Server para crear el dialogo, el servidor puede poner un dialogo sobre el UI o cualquier aplicación que esté en primer plano.

Comunicación usando OBEX

Proporciona un método para la transferencia de un dispositivo a otro.
Corre sobre diferentes medios de transporte, incluyendo IrDA y bluetooth RFCOMM.
OBEX es adecuado para operaciones sencillas como la transferencia de un archivo.
Se tratará RFCOMM como medio de transporte específico para bluetooth.
Está implementado en estas clases:

- **TObexBluetoothProtocolInfo**
- **TObexConnectInfo**
- **MObexServerNotify** => Incluidas en obex.h y definidas en las cabeceras de obexXXX.h
- **CObexClient**
- **CObexServer**

Una aplicación puede actuar como cliente o como servidor.

Tendremos un servidor bluetooth en el lado del sevidor y un dispositivo bluetooth y un servicio descubridor en el lado del cliente, de tal forma que el servidor publica un servicio bluetooth, el cliente lo descubre y se conecta a el para posteriormente poder mandar objetos OBEX al servidor. El ejemplo se implementa usando la clase CObexFileObject.

Definiendo el protocolo Bluetooth.

Para definir el protocolo usaremos la clase TObexProtocolInfo y mas concretamente una clase que hereda de ella que es **TObexBluetoothProtocolInfo** .

En esta clase tendremos dos atributos:

- **TRfcommSockAddr iAddr** → representa el canal o numero de puerto para
  conectarnos al dispositivo remoto.

- **TBuf<60> iTransport** → permite a la aplicación cliente especificar un string con el protocolo que se va a usar.

Objetos OBEX

Todos los datos transmitidos están recubiertos en un objeto contenedor antes de ser enviados. Hay tres importantes tipos de objetos OBEX. Todos heredan de CObexBaseObject. Podemos ver el API de las clases en el archivo cabecera de obex.h.

- **CObexFileObject** → para mandar archivos sobre OBEX.
- **CObexBufObject** → cuando enviamos un trozo de memoria.
- **CObjectNullObject** → para un objeto vacío.

Mecanismo de notificación del servidor OBEX

MObserverNotify es una clase observadora de Symbian OS. Se usa por el sistema operativo para informar a la aplicación cliente de eventos de comunicación OBEX. Típicamente un OBEX Server implementaría esta clase.

MObexServerNotify observer class define todas sus funciones como privadas con lo que sólo puede ser usada como observadora con la clase CObexServer.

En varios ejemplos de OBEX, CObjetExchangeServer implementa la interfaz MObexServerNotify.

Estas son algunas de las funciones:

- ErrorIndication
- TransportUpIndication
- TransportDownIndication
- ObexConnectIndication
- ObexDisconnectIndication
- PutRequestIndication
- PutPacketIndications
- PutCompleteIndication
- GetRequestIndication
- GetPacketIndication
- GetCompleteIndication
- AbortIndication

OBEX client

CObexClient provee a una aplicación cliente con la funcionalidad necesaria para transmitir, request a un OBEX Server.

Cuando se construye se le debe especificar la dirección del dispositivo del servidor y el protocolo que se usa para conectarse, esta información esta encapsulada en un objeto TObexBluetoothProtocolInfo.

Funciones: Connect (conectar), Get (request), Put (enviar un objeto OBEX).

OBEX Server

CObexServer permite a una aplicación cliente ofrecer servicios OBEX a otros dispositivos. Cuando construimos ObexServer la aplicación cliente tiene que suministrar TObexProtocolInfo, esta proporcionará información de escucha al servidor.
La información incluye el protocolo y el puerto.

Para arrancar el servidor usaremos la función **Start,** una vez hecho esto el cliente debe proporcionar una instancia de la clase observadora MObexServerNotify. El sistema operativo, vía CObexServer informará a la observadora de cualquier evento.

La función **Stop** parará el servidor y para saber el estado del servidor podemos usar la funcion **IsStarted** y **CurrentOperation**.

Un modo fácil de usar OBEX con UI

Send UI es un API adecuado que oculta los detalles del portador al desarrollador. Puede ser usado fácilmente para enviar SMS, MMS, e-mail y también para transferir sobre Bluetooth o infrarrojos.

El procedimiento básico para usar Send UI es añadir el comando Send a las opciones de menú. El comando Send muestra la lista de los posibles portadores, entre los que se puede encontrar bluetooth. Esta lista puede variar dependiendo del dispositivo móvil y su configuración.

Después de seleccionar el portador, el envió empezará. En el caso de que el portador sea bluetooth, el Plug-in notificador de selección de dispositivo bluetooth es llamado y la lista de dispositivos es mostrado después de que el usuario haya seleccionado el elemento de menú. El archivo seleccionado será enviado usando el protocolo OBEX al dispositivo seleccionado.

En S60 3rd Edition, Send UI API ha sido eliminada, el nuevo APi es menos complejo y mas fácil de usar. La clase CSendAppui ha sido eliminada y CsendUI se ha introducido en la cabecera de Sendui.h. Todos los valores MTM (Message Type Module) están definidos en SenduiMtmUids.h (KsenduiMtmBtUidValue, el valor de tipo mensaje) de Bluetooth es 0x10009ED5).

Ejemplo de uso de Send UI API.

Pasos a seguir para el uso de SendUI API:

1. Introducir un objeto CSendUi (recomendable declarar CSendUI como variable miembro e inicializarla en el constructor).

2. Añadir el Send Menu item.

3. Mostrar el Send list Query (o cascade menú en S60 1st and 2nd edition) y crear el mensaje para enviar.

En el ejemplo OBEX del forum nokia podemos ver como usar SendUI API. Los cambios en la tercera edicion pueden tomarse en consideración usando macros.

```
#ifdef __SERIES60_3X__
iSendUi = CSendUi::NewL();
#else
iSendUi = CsendAppUi::NewL(EsendUi);
#endif
```

CsendAppUi reuqiere el valor del comando Send menu item para pasarlo como argumento, ya que las funciones de envío de esta clase reciven el ID reflejando lo que el usuario ha seleccionado (bluetooth…). Sabiendo el valor del

comando Send menú item podemos calcular que portador se ha seleccionado contando el offset.

En 3rd edition, la clase CSendUi tiene la función AddSendMenuItemL que añada el elemento de menú. Las opciones deseadas del medio (bletooth, IrDA..) pasando también como argumento TSendingCapabilities.

```
void CSendUi::AddSendMenuItemL ( CEikMenuPane &
aMenuPane,
TInt aIndex,
TInt aCommandId,
TSendingCapabilities aRequiredCapabilities =
KCapabilitiesForAllServices
)
```

aMenuPane el un menu pane en el que el Send menú item ha sido añadido

aIndex es el lugar del Send menu item.

aRequiredCapabilities son las características requeridas por el MTMs para ser mostradas.

El siguiente fragmento de código, tomado del ejemplo de OBEX del forum nokia, demuestra como usar el Send UI API para enviar contenido como adjuntos. Si no hay tipos de mensajes disponibles que casen con TSendingCapabilities, entonces El Send menú item no se añade al menú pane.

```
void CBTObjectExchangeAppUi::DynInitMenuPaneL( TInt
aResourceId, CEikMenuPane* aMenuPane )
{
    if ( aResourceId == R_BTOBJECTEXCHANGE_MENU )
    {
    //...
    // Display the Send menu item (if there are bearers)
    // Show only the bearers which support attachments
    TSendingCapabilities capabilities(0,0,
    TSendingCapabilities::ESupportsAttachments );
    #ifdef __SERIES60_3X__
    //In S60 3rd Edition the menu item is not created in the rss
    TInt position = 0;
    //so find a desired command
    aMenuPane->ItemAndPos(
    EBTObjectExchangeClearList, position );
    //and add the Send menu item next to it
    iSendUi->AddSendMenuItemL(*aMenuPane, position-1,
    ESendViaSendUi, capabilities );
    #else
```

```
        //In S60 1st and 2nd Edition we have the Send menu item
        // with its cascade menu defined in the rss file
        TInt position = 0;
        aMenuPane->ItemAndPos( ESendViaSendUi, position );
        iSendUi->DisplaySendMenuItemL (*aMenuPane, position,
        capabilities );
    aMenuPane->DeleteMenuItem(ESendViaSendUi);
        #endif
        }
```

Version 2.0 | December 22, 2006 S60 Platform: Bluetooth API Developer's Guide | 56

```
        #ifndef __SERIES60_3X__ //this is not needed in 3rd ed.
        // Show the SendUI cascade menu
        else if ( aResourceId == R_BTOBEX_SENDUI_MENU)
        {
        iSendUi->DisplaySendCascadeMenuL(*aMenuPane, NULL);
        }
        #endif
        }
    }
```

El uso de CSendUi en la tercera edición requiere menor trabajo para el programador que se usamos CSendAppUi. Con CSendUi el elemento del menú no necesitan introducir en el .rss el elemento de menú.
Después de que el elemento de menú es visible y que el usuario selecciona el comando del menú, el tratamiento se hace en HandleCommandL:

```
    void CBTObjectExchangeAppUi::HandleCommandL( TInt
    aCommand )
    {
    //...
    switch ( aCommand )
    {
    //...
    case ESendViaSendUi:
    //These are the commands from Send cascade menu
    //(actually needed only in 1st and 2nd Edition)
    case ESendViaSendUi1: //FALLTHROUGH
    case ESendViaSendUi2: //FALLTHROUGH
    case ESendViaSendUi3: //FALLTHROUGH
    case ESendViaSendUi4: //FALLTHROUGH
    case ESendViaSendUi5: //FALLTHROUGH
    case ESendViaSendUi6: //FALLTHROUGH
    case ESendViaSendUi7: //FALLTHROUGH
    case ESendViaSendUi8: //FALLTHROUGH
    case ESendViaSendUi9:
```

```
{
//With all the commands we send a file.
//Pass the command to the function. It's required
//by CSendAppUi sending function
SendFileViaSendUiL(aCommand);
break;
}
//...
}
```

En la tercera edición el comando ESendViaSendUi es el único comando del que hay que encargarse, en la primera y segunda edición usan menú de cascada y hay más comandos a tratar, de hecho tantos como medios (bearers) tengamos. Como es difícil saber cuantos vamos a tener es recomendable poner un numero suficiente de comandos para tratar los diferentes medios (bearers).

En el ejemplo OBEX después de que el usuario ha seleccionado el comando de menú Send, se llama a la función SendFileViaSendUiL. Aunque no es una función del API de OBEX, es simplemente una función de ayuda en el ejemplo.

Con CSendAppUi, el ID del comando se necesita en el envio poruq describe el medio seleccionado (bearer), Hay una diferencia fundamental entre CSendUi y CSendAppUi, con este ultimo el bearer se selecciona cuando HandleCommandL es ejecutada. Con CSendUi, la selección esta hecha desde la lista de bearer.

Seguridad y Configuración de Bluetooth en Symbian

Gestor de Seguridad (Security Manager) Bluetooth

El Security Manager maneja la seguridad de las conexiones bluetooth. Permite al cliente especificar unas preferencias de seguridad cuando se establece una conexión:

- autorización de usuario
- autentificación
- encriptación

Estas preferencias son aplicadas a un servidor específico, con un protocolo y un canal. Este gestor asegura que dicha conexión se realiza con estas preferencias.

Dichas preferencias se realizan de distinta manera según la versión del API:

Bluetooth API v1:

- RTBSecuritySettings::RegisterService

Bluetooth API v2:

- TBTSockAddr::SetSecurity método que se usa cuando un socket en escucha se abre.
- RBTSecuritySettings::UnregisterService método que se llama para borrar todos las preferencias sobre la seguridad cuando un socket se cierra.

El gestor esta implementado como un servidor del API de Symbian.

La información de la seguridad de la conexión bluetooth se encuentra encapsulada en la clase TBTServiceSecurity definida en btmanclient.h en el bluetooh API v1 y en bt_sock.h en posteriores arquitecturas.

Dichas opciones de seguridad se configuran con valores booleanos (true/false). SetAutehentication, SetAuthorisation y SetEncryption son de esta clase.
Para denegar el acceso la función setDenied. El cliente debe de especificar el servicio, protocolo y el canal en donde aplicar dicha configuración. Cada servicio bluetooth está identificado con un UUID y cada protocolo con un número entero. Hay una lista de valores de protocolos y servicios en bt_sock.h. Entre las diferentes versiones hay cambios con respecto a estos valores por eso hay que comprobar antes en el SDK que se esta usando.

Servicio de seguridad en la arquitectura Bluetooth API v2

Esta arquitectura se encuentra desde la version de Symbian 8.0 o posteriores. Los valores de las preferencias de seguridad son borradas automáticamente al cerrarse el socket. Se configura con el metodo TBTSockAddr::SetSecurity (antes hay que usar el RBTSecuritySettings:RegisterService).

```
//solicitamos el recurso socket
User::LeaveIfError(iListener.Open(iSockServ, _L("L2CAP")));
TL2CAPSockAddr addr;

//configuramos su puerto
addr.SetPort(KSDPPSM);
TBTServiceSecurity sdpSecurity;

//configuramos la seguridad (sin seguridad)
sdpSecurity.SetUid(KUidServiceSDP);
sdpSecurity.SetAuthentication(EFalse);
sdpSecurity.SetAuthorisation(EFalse);
sdpSecurity.SetEncryption(EFalse);
```

```
sdpSecurity.SetDenied(EFalse);

//la asignamos
addr.SetSecurity(sdpSecurity);
User::LeaveIfError(iListener.Bind(addr));
User::LeaveIfError(iListener.Listen(iQueSize));


//también podíamos haber configurado seguridad con autentificación de la siguiente manera

TBTServiceSecurity sec;
sec.SetAuthentication(ETrue);
TBTSockAddr sockaddr;
sockaddr.SetBTAddr(aDevice.BDAddr());
sockaddr.SetPort(aChannel);
sockaddr.SetSecurity(sec);

//una vez autenficado se puede conectar.

sock.Connect(sockaddr,iStatus);
```

Servicios de seguridad en arquitectura bluetooh APIv1

Este servicio funciona como un servidor de Symbian, si una aplicación cliente deseara conectarse tiene que pasar por dicha seguridad.
RBTMan proporciona la conexión con el gestor de seguridad (definida en btmanclient.h) y RBTSecuritySettings proporciona la funcionalidad de seguridad (definida en btmanclient.h)

Con RegisterService y UnregisterSevices especifica y borra, respectivamente las preferencias de la seguridad una vez establecida la conexión con el gestor de seguridad.
Dichas funciones son asíncronas y requieren que la aplicación cliente proporcione un objeto estado (TRequestStatus). El servidor utiliza dicho objeto para notificar a la aplicación cliente que ha sido completada su intento de modificar los requerimientos de seguridad. El objeto TRequestStatus contiene un entero que representa si ha sido satisfactorio o no el intento.
Pasos para configurar la seguridad en una conexión bluetooth:

- Conectar con el servidor de seguridad y abrir una sesión

```
RBTMan secManager;
```

RBTSecuritySettings secSettingsSession;
User::LeaveIfError(secManager.Connect());
CleanupClosePushL(secManager);
User::LeaveIfError(secSettingsSession.Open(secManager));
CleanupClosePushL(secSettingsSession);

- Cambiar las preferencias de seguridad con un objeto TBTServiceSecurity

  TBTServiceSecurity serviceSecurity(KUidBTObjectExchangeApp, KSolBtRFCOMM, 0);
  serviceSecurity.SetAuthentication(aAuthentication);
  serviceSecurity.SetEncryption(aEncryption);
  serviceSecurity.SetAuthorisation(aAuthorisation);
  serviceSecurity.SetChannelID(aChannel);

  (En este ejemplo se aplica las preferencias de seguridad a un servicio OBEX utilizando un protocolo de transmisión RFCOMM y el canal, autentificación, autorización y encriptación han sido pasados por parámetro)

- Registrar dichas preferencias

  TRequestStatus status;
  secSettingsSession.RegisterService(serviceSecurity, status);

- Esperamos a dichas preferencias se hayan recibido

  User::WaitForRequest(status);
  User::LeaveIfError(status.Int());

- Cerrar la sesión y la conexión

  ```
  CleanupStack::PopAndDestroy(2);
  ```

Publish & Subscribe Bluetooth

La configuración dinámica del bluetooth esta manipulada por la nueva API de Publicadores y subscriptores de la librería bt_subscribe.h (hay cambios clave en SDK S60 3rd Edición, consultar la tabla 5)

Categorías y propiedades clave de bluetooth

La pila de los valores de las propiedades bluetooth publicadas se engloban en una categoría bluetooth.
Una aplicación puede recuperar, registrar y cambiar el valor de una propiedad. Pero no pueden acceder a las propiedades clave. Para suplir esto pueden crear una petición para cambiar el valor de la propiedad equivalente en la categoría Bluetooh Control. El valor de la clave en la categoría será actualizada cuando la solicitud sea atendida.

Por tanto un patrón de una aplicación que desea cambiar una preferencia es el siguiente:

- Definir una clave en la categoría Bluetooth Control.
- Suscribir la clave deseada en la categoría (KPropertyUidBluetootCategory)
- Llamar al metodo RProperty:Set() en la categoría Bluetooth Control (KPropertyUidBluetootControlCategory)
- Cuando la pila de solicitudes esta servida, se "re-plublicará" el valor de la clave sea o no cambiado. Llamando al RunL de sus subscriptores.

Para recibir una propiedad se hace llamando a RPrperty::Get().
Las propiedades clave disponibles estan en la tabla 5.

Key	Type	Description
KPropertyKeyBluetoothLocalDeviceAddress S60 3rd Edition: **KPropertyKeyBluetoothGetLocalDeviceAddress**	Byte Array	Dispositivo de dirección local como descriptor
KPropertyKeyBluetoothPHYCount S60 3rd Edition: **KPropertyKeyBluetoothGetPHYCount**	Integer	Número actual de conexiones bluetooth físicas
KPropertyKeyBluetoothConnecting Anticuada en S60 3rd Edition, se puede usar la instancia **KPropertyKeyBluetoothGetConnectingStatus**.	Integer	1 – Si esta buscando otro dispositivo 0 – Si no esta buscando otro dispositivo
KPropertyKeyBluetoothScanning Anticuada en S60 3rd Edition, se puede usar la instancia **KPropertyKeyBluetoothGetScanningStatus**. Hay tambien una constante **KPropertyKeyBluetoothSetScanningStatus** añadida en S60 3rd Edition.	Integer	El estado actual del escaneo

KPropertyKeyBluetoothLimitedDiscoverable No existe en la versión S60 3rd Edition.	Integer	Valores del enumerado: **THCIScanEnable**: **ENoScansEnabled**, **EInquiryScanOnly**, **EPageScanOnly** or **EinquiryAndPageScan**.
KPropertyKeyBluetoothDeviceClass No esta en la version S60 3rd Edition.	Integer	El entero que representa la clase de dispositivo
Key **KPropertyKeyBluetoothRegistryTableChange** Esta anticuada en la versión S60 3rd Edition.	Integer	El índice de la tabla que ha cambiado.

Tabla5

Definir claves en la categoría Bluetooth Control

Si la aplicación quiere cambiar algunas propiedades debe definirlas antes en la categoría Bluetooth Control.

```
TIntr=iProperty.Define(KPropertyUidBluetoothControlCategory,
KPropertyKeyBluetoothScanning,
RProperty::EInt);
if((r!=KErrAlreadyExists))
{
Use::LeaveIfError(r);
}
```

Después de esto el valor puede ser cambiado llamando a RProperty::Set().

Cuando una aplicación quiere obtener algunos valores de BlueTooth Publish and Susbscribe se utiliza el metodo RProperty::Get() (Siendo la instancia RProperty utilizada en el ejemplo de arriba):

```
TPckgBuf<TBTDevAddr> aDevAddrPckg;
//IN V1:
TInt error = RProperty::Get(KPropertyUidBluetoothCategory,
KPropertyKeyBluetoothLocalDeviceAddress, aDevAddrPckg);
//IN V2:
TInt error = RProperty::Get(KPropertyUidBluetoothCategory,
KPropertyKeyBluetoothGetLocalDeviceAddress, aDevAddrPckg);
```

En la ayuda del s60 SDK hay mas informacion sobre Publish and Subscribe.
Para probar si el Bluetooth es soportado hay que usar CFeatureDiscorvery (featdiscovery.h y featureinfo.h) en S60 2nd Edicion.

```
TBool isSupported =
CFeatureDiscovery::IsFeatureSupportedL(KFeatureIdBt);
```

Para mirar el estado de la energia BlueTooth en el SDK 3rd edición se utiliza el Central Repository API:

```
CRepository* crep = CRepository::NewL(KCRUidBluetoothPowerState);
TInt value=0;
User::LeaveIfError( crep->Get(KBTPowerState, value) );
```

La clase CRepository esta definida en centralrepository.h y las constantes **KCRUidBluetoothPowerState y KBTPowerState en BTServerSDKCRKeys.h**

En la 2nd Edicion está CSettingInfo (settinginfo.h y settinginfoids.h para mirar el estado de la energia.

```
#include <SettingInfo.h>
#include <aknnotewrappers.h> //NULL because there's no observing
functions used
CSettingInfo* settingInfo = CSettingInfo::NewL(NULL);
CleanupStack::PushL(settingInfo); //getting the current bluetooth power
mode TInt btPowa; TInt err = settingInfo->Get(
SettingInfo::EBluetoothPowerMode, btPowa ); if (err)
{ //handle error
}
if (btPowa)
{
//power is on
}
else {
//power off // Do something to switch BT on, // e.g. show notifier, see next
chapter
} CleanupStack::PopAndDestroy(settingInfo);
```

Si el dispositivo esta iniciado usando Bluetooth UI, el estado de la batería es chequeado automáticamente, y si el Bluetooth esta apagado el usuario proporciona con un dialogo que lo encienda. Esto no es automático en todos los casos: desde la aplicación servidor siempre esta al tanto de chequear la batería, en el caso del cliente si RHostResolver esta usando el dispositivo.

El chequeo de batería bluetooth puede estar usando Bluetooth Notifier API. Este API se introduce en el SDK 2nd edición serviPack1 (btnnotifierapi.h en 3rd edicion y btnotif.h en 2nd edición SDK's FP1, FP2 y FP3)

Antes la 2nd edición servipack 1 el valor constante ha de ser usado directamente.

```
const TUid KPowerModeSettingNotifierUid = {0x100059E2}
```

Para Mostar el dialogo, usar la siguiente funcion:

```
StartNotifierAndGetResponse(TRequestStatus &aRs, TUid
aNotifierUid, const TDesC8 &aBuffer, TDes8 &aResponse);
```

NCF 1.2 (Nokia Conectivity Framework)

El NCF 1.2 (**Nokia** Conectivity Framework) es un software específico para probar Sw para dispositivos móviles Nokia con Symbian incorporado. Con esta aplicación se podría simular la comunicación que realizan dos teléfonos móviles. La realidad es que después de mucho buscar y mucho probar no se ha conseguido hacer que funcione correctamente, solo hubo un momento que pareció funcionar pero enseguida dejo de hacerlo.

Se ha investigado el programa y se ha comprobado que los emuladores no funcionan como debieran en el tema de las comunicaciones.

A continuación detallamos los pasos que deberían servir para hacerlo funcionar y los problemas que nos encontramos y las soluciones a algunos de ellos que conseguimos solventar.

Problemas de instalación NCF 1.2

1. Al no tener el JRE 1.4.1_02 se instalaba bien pero luego no conseguía hacerlo funcionar. Se solvento instalando el JRE correcto.

2. Una vez instalado, intente descargar el SDK 3ª edición FP1 de la pagina de forum.nokia.com pero no me descargaba bien con lo cual no he podido instalar ese SDK, y he instalado el SDK 3ª edición que nos pasó Pepe en el curso.

3. Con el SDK 3ª edición no he conseguido arrancar dos emuladores a pesar de poner en la propiedad de instante support que fuera múltiple, con el SDK 2.0 si que me permite emular varios, aunque no me funciona en el mismo puerto y cambiando los puertos no me funcionan los ejemplos tampoco, a veces pone error y otras veces se queda buscando dispositivos todo el rato.

4. Lo único que he conseguido ha sido arrancar varios emuladores pero con la versión del SDK 2.0, y los ejemplos de bluetooth como el de las bolitas no me funcionaban. En mis intentos se quedaba buscando dispositivos y se quedaba así todo el rato. Al hacerlo Javi le daba error directamente al ejecutar la conexión bluetooth en el ejemplo de las bolitas (y todo esto con el SDK 2.0 porque con el otro no ejecutaba dos emuladores.)

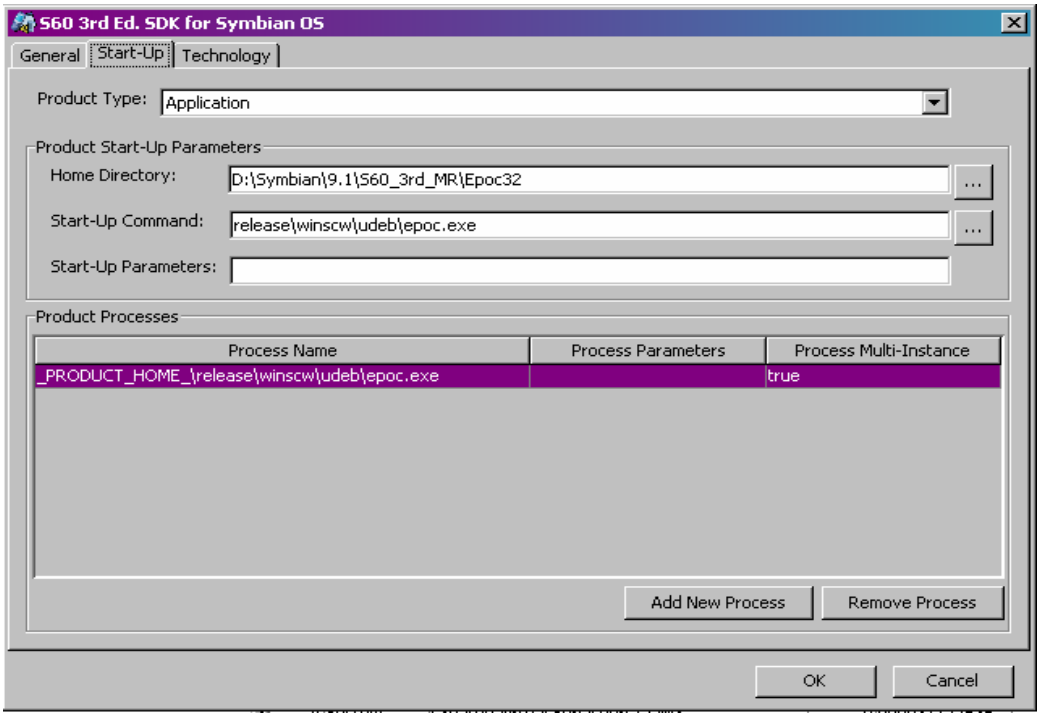
Pasos que se han seguido en las pruebas de la instalación:

1. Desinstalar los JRE.

2. Instalar 1.4.1_02 o superior. (puede que sea necesario reiniciar)

3. Instalar el NCF 1.2.

4. Se supone que al arrancar el NCF 1.2 detecta automáticamente los SDK que tengas instalados. El SDK 3ª edición si que me lo detectó pero el 2.0 no y para configurarlo manualmente hay que crear un nuevo producto: Tool-> Create New Product Integration. Luego le das un nombre y eliges las tecnologías que quieras que tenga en la pestaña Tecnology. Y en la pestaña de Start-up hay que configurarlo de la siguiente manera:

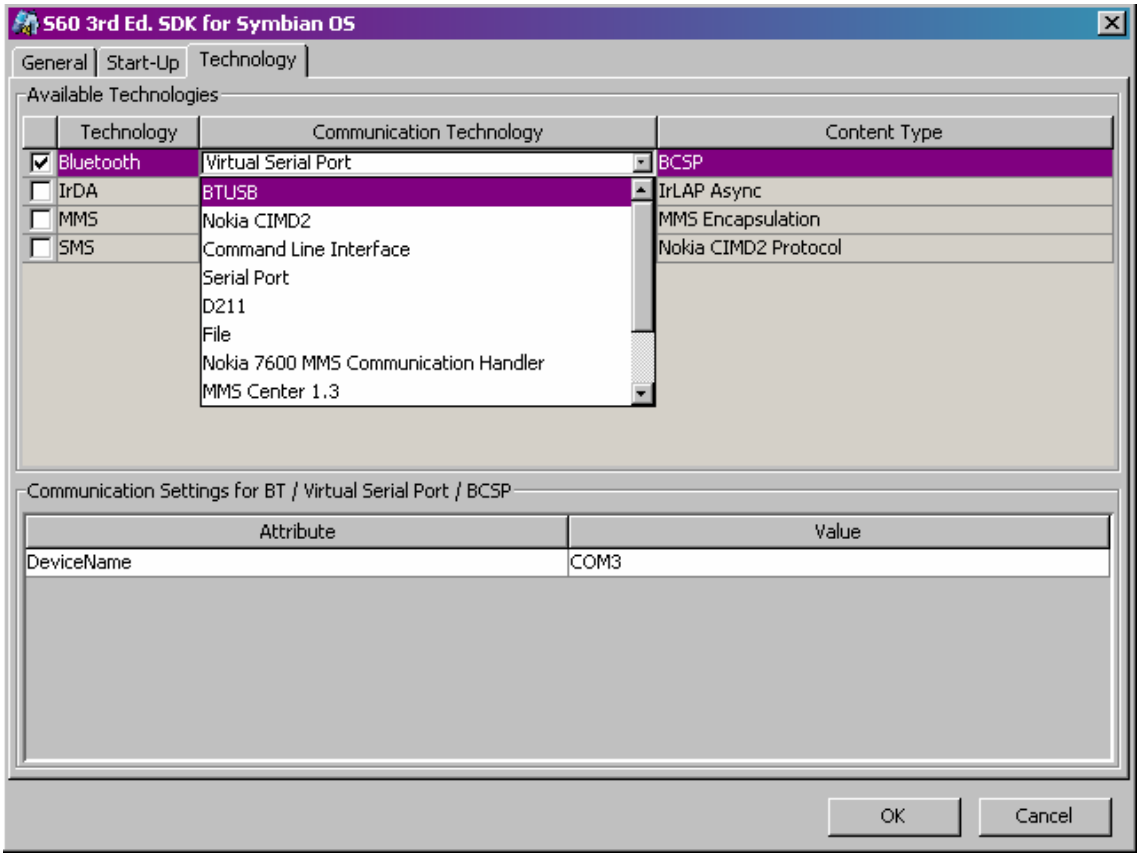


Esta configuración es para la 3ª edición, y dependerá de la unidad (C, D,…) donde tengamos instalado el SDK.

Si queremos que podamos poner múltiples instancias tenemos que poner en la pestaña General, en la propiedad Instante support, la opción múltiple.

Con esto podríamos en teoría usar el NCF para arrancar emuladores, y si hemos puesto en la pestaña tecnología la opción de bluetooth podríamos usar el bluetooth.

El problema ahora viene en saber como configurar el bluetooth, ya que tiene varias opciones y probando varias de ellas no hemos conseguido hacerlo funcionar correctamente. En la siguiente imanen podemos ver las algunas de las opciones que nos dan.



Con la primera que aparece, BTUSB, decía que no hay hardware para usar esta opción y que si queríamos instalarlo, pero como no teníamos en Hw no lo hemos podido probar.

Lo único que parecía que podía funcionar era la opción de Virtual Serial Port, que nos instalaba un puerto serie virtual. Cuando lo emulamos, dice que tienen el mismo puerto los dos emuladores y tenemos que poner cada emulador en un puerto. Y como he dicho antes al ejecutar el ejemplo se queda buscando dispositivos y no sale de ahí.

No se si con el Hw que nos pide en la opción de BTUSB funcionaría.

Resumen de los problemas:

- No arranca el programa si no tienes el JRE correcto.
- Con el SDK 2.0 puedes hacer múltiples instancias del emulador configurando el bluetooth con la opción de virtual serial port, aunque no me funcionan los ejemplos (se queda buscando dispositivos continuamente) y además no me permite que tengan el mismo puerto.
- Con el SDK 3ª edición no me deja instanciar múltiples emuladores, solo me arranca un emulador.

Conclusiones

El NCF 1.2 sería muy útil para probar el software de comunicaciones que desarrolláramos ya que si no se dispone de teléfonos físicos donde probarlo no podríamos verificar el funcionamiento de los programas.

El problema es que al no funcionar correctamente no podemos probar el software de otro modo que no sea en el dispositivo móvil, con lo cual hacia mas laboriosa la tarea de verificar el programa.

El objetivo que se marco sobre el desarrollo de un sniffer, no se ha podido completar debido a que necesitábamos configurar la hardware bluetooth en modo promiscuo para que recogiera todos los paquetes que circularan por la red bluetooth y esto esta fuera de nuestro alcance. Esto es debido a que por la seguridad del sistema operativo Symbian, no tenemos acceso a esta configuración, siendo Symbian quien debiera modificarlo.

Debido a esta barrera que nos impedia avanzar en el desarrollo del sniffer se cambió la linea de investigación y se opto por ver formas de vulnerar la seguridad de los dispositivos bluetooth que es lo que pasamos a detallar a continuación.

Revisión del ataque Blue Mac Spoofing

Introducción

La mayoría de usuarios de teléfonos móviles Bluetooth ha tenido la necesidad alguna vez de emparejar su teléfono con otro dispositivo con el fin de transferir archivos vía Bluetooth o conectarse a equipos manos libres o receptores GPS. Es un hecho que la conducta habitual suele ser mantener esos enlaces activos aun cuando estos se encuentren en desuso o la conexión haya sido esporádica.

¿Qué ocurriría si cada uno de esos enlaces activos pudiera convertirse en una puerta trasera a nuestro teléfono, con acceso transparente al control total de las funciones del teléfono y los archivos almacenados? En efecto, dado que todos los mecanismos de seguridad empleados por Bluetooth se realizan a nivel de dispositivo, y no de usuario, suplantar la identidad de un dispositivo emparejado y utilizar sus credenciales de confianza para acceder a un teléfono sin que el usuario se percate resulta una acción trivial. En esto consiste el ataque Blue MAC Spoofing.

El ataque Blue MAC Spoofing, al contrario que cualquier ataque publicado hasta la fecha que afecte a teléfonos móviles Bluetooth, no explota una implementación incorrecta de Bluetooth por parte de los fabricantes, sino un fallo del estándar en sí mismo. Se trata de una vulnerabilidad intrínseca a los mecanismos de seguridad utilizados por Bluetooth y, por ello, no sólo afecta a todos los teléfonos móviles Bluetooth, sino a cualquier equipo que haga uso de esta tecnología de comunicaciones inalámbricas.

Mecanismos de Seguridad en Bluetooth

A fin de poder entender el desarrollo del ataque, es importante conocer los mecanismos de seguridad que utiliza Bluetooth y porque son fácilmente explotables debido a su diseño.

- **Autenticación**

Es el proceso por el cual un dispositivo Bluetooth verifica su identidad en otro dispositivo para poder acceder a los servicios que ofrece.

La primera vez que dos dispositivos intentan comunicarse, se lanza un procedimiento de inicialización denominado emparejamiento para crear una clave de enlace común de una forma segura. El procedimiento estándar requiere que el usuario de cada dispositivo introduzca un código de seguridad Bluetooth,

conocido como PIN, de hasta 16 bytes de longitud que debe ser el mismo en ambos casos. A partir de este código PIN Bluetooth, la dirección BD_ADDR de cada dispositivo y varios números aleatorios de 128 bits se obtiene la clave de enlace común a ambos dispositivos a través de los algoritmos E22 y
E2 1.

Una vez que los dispositivos emparejados disponen de la clave de enlace, utilizan esta clave común para autenticarse automáticamente en sucesivas conexiones. El proceso de autenticación está basado en un esquema de desafío/respuesta que comprueba la identidad del dispositivo que intenta conectarse en el destino.

- **Autorización**

Es el procedimiento que determina los derechos que tiene un dispositivo Bluetooth para acceder a los servicios que ofrece un sistema.

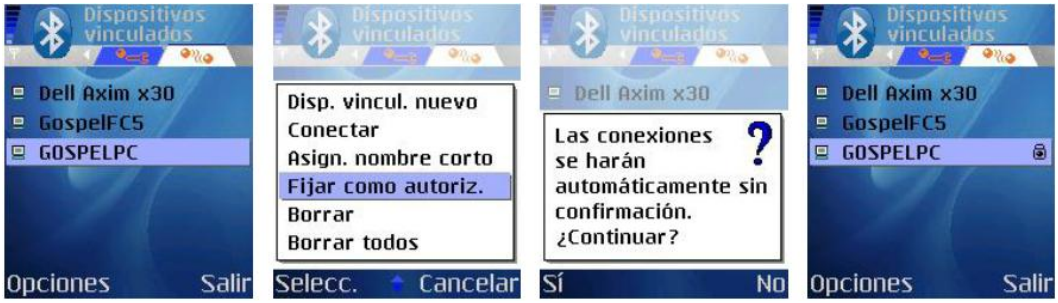
El mecanismo de autorización en dispositivos Bluetooth se lleva a cabo mediante niveles de confianza y se gestiona mediante una lista de dispositivos de confianza que determina la capacidad de acceso a los servicios: total, parcial o restringida y nula.

- Un dispositivo de confianza dispone de acceso sin restricciones a todos los servicios.
- Un dispositivo no confiable dispone de acceso restringido a uno o varios servicios o incluso no se le permite el acceso a ningún servicio.

Todo dispositivo Bluetooth que implemente servicios con autorización dispone de una base de datos interna con una lista de dispositivos de confianza y que tiene el siguiente formato:

Campo	**Estado**	**Contenido**
BD_ADDR	Obligatorio	Dirección MAC del dispositivo
Nivel de confianza	Obligatorio	De confianza/No de confianza (Booleano)
Clave de enlace	Obligatorio	Clave de enlace
Nombre	Opcional	Nombre del dispositivo
Class of Device	Opcional	Identificador de la clase de dispositivo

En la mayoría de teléfonos móviles es habitual solicitar automáticamente al usuario si desea marcar un dispositivo como autorizado justo después de haberse emparejado con el mismo, aunque también se añade la opción de hacer esto posteriormente de forma manual. Por lo tanto, es posible considerar que un dispositivo emparejado es también un dispositivo autorizado.



En el caso de que un dispositivo de confianza intente acceder a un servicio autorizado, no se requiere ningún procedimiento de confirmación, accede de forma transparente.

En el caso de que un dispositivo no confiable intente acceder a un servicio restringido, se requiere un procedimiento de confirmación explícito al usuario para permitir o denegar el acceso de ese dispositivo a ese determinado servicio.

La realidad de los mecanismos de Seguridad en Bluetooth

Tras la publicación de los primeros ataques a teléfonos móviles Bluetooth, en especial el ataque Bluesnarf (extracción de archivos de un teléfono móvil Bluetooth a través del Perfil de Carga de Objetos, OBEX Object Push) sin autorización del usuario propietario), los fabricantes tomaron mayor conciencia de la necesidad de fortificar el acceso a los perfiles Bluetooth que soportaban sus dispositivos manufacturados.

Se decidió que, en adelante:

- Todos los perfiles Bluetooth debían requerir autenticación, a excepción de aquellos perfiles cuya funcionalidad quedaría limitada por el modelo de uso. Por ejemplo: el perfil de Carga de Objetos utilizado para transferencia simple y espontánea de archivos (tarjetas de visitas, contactos, etc.) y marketing de proximidad.

- Todos los perfiles Bluetooth debían requerir autorización, con la posibilidad de que aquellos dispositivos ya emparejados fueran autorizados automáticamente o pudieran ser marcados como autorizados por el usuario.

Con esta medida, el escenario de ataque a teléfonos móviles se transformó. Dado que la implementación de los fabricantes iba a ser robusta, siguiendo las

recomendaciones del estándar Bluetooth, la nueva estrategia es atacar al estándar directamente, ¿cómo saltarse los mecanismos de seguridad utilizados por Bluetooth?

Si nos tomamos un tiempo para estudiar detenidamente la arquitectura de seguridad en Bluetooth, salta a la vista la simplicidad de su diseño y lo débil que resulta.

Authorisation and Authentication

Authorisation is the process of deciding if device X is allowed to have access to service Y. This is where the concept of 'trusted' exists. Trusted devices (authenticated and indicated as "trusted"), are allowed access to services.

Security Levels of Services

Authorisation Required: Access 15 only granted automatically to trusted devices (i.e., devices marked as such in the device database) or untrusted devices after an authorisation procedure.

Fuente: Apdo. 3.2 Security Levels — Bluetooth SecurityArchitecture Version 1.0 (www.bluetooth.com)

Esto significa que un dispositivo autorizado puede acceder automáticamente a todos los servicios que requieran autorización. Para que un dispositivo sea considerado como autorizado debe estar marcado como tal en la lista de dispositivos de confianza y para ello debe tratarse también de un dispositivo autenticado (recordemos que el campo clave de enlace es obligatorio en la base de datos de dispositivos de confianza).

Sin embargo, un hecho importante que se omite es que si un servicio no requiere autenticación, la clave de enlace no entra en juego necesariamente y el acceso se basa únicamente en la dirección BD_ADDR de dispositivo, de forma que si existe en la lista de dispositivos de confianza, este queda autorizado.

Authentication

The authentication procedure is based on a challenge-response scheme [..]. The verifier sends [...] a random number (the challenge) to the claimant. The claimant calculates a response, that is a function of this challenge, the claimant's BD_ADDR and a secret key. The response is sent back to the verifier, that checks if the response was correct or not. [...] A successful calculation of the authentication response requires that two devices share a secret key.

Fuente: Apdo. 4.2 Security - Core v2.O + EDR (www.bluetooth.org, disponible para miembros del SIG)

Esto significa que la autenticación se basa en la dirección BD_ADDR de dispositivo y en la clave de enlace secreta compartida, de forma que si el esquema desafío/respuesta es positivo, este queda autenticado.

El esquema desafío/respuesta de autenticación transcurre de la siguiente forma:

1) El dispositivo reclamante envía su dirección BD ADDR al dispositivo verificador.

2) El verificador devuelve un desafío aleatorio de 128 bits al reclamante.

3) El reclamante usa el algoritmo El para generar la respuesta de autenticación (SRES) de 32 bits, usando como parámetros de entrada la dirección BD_ADDR del reclamante, la clave de enlace almacenada y el desafío recibido. El verificador realiza la misma operación en paralelo.

4) El reclamante devuelve la respuesta SRES al verificador.

5) El verificador comprueba la respuesta SRES recibida por el reclamante con la respuesta SRES calculada por él.

6) Si los valores de SRES coinciden, el verificador autentica al reclamante.

En resumen,

- La autorización se basa únicamente en la dirección BD_ADDR de dispositivo.

- La autenticación se basa en la dirección BD_ADDR de dispositivo y en la clave de enlace secreta compartida.

Dado que tanto el mecanismo de autorización como el esquema de desafío/respuesta utilizado para la autenticación verifican la identidad de dispositivos, no usuarios, surgen dos preguntas clave:

- ¿Qué pasa si un equipo atacante suplanta la dirección BD_ADDR de un dispositivo de confianza? ¿Queda autorizado en el dispositivo destino?

- ¿Qué pasa si un atacante tiene acceso a la clave de enlace de uno de los dispositivos emparejados? ¿Puede utilizarla para autenticarse en el otro dispositivo?

La respuesta es **sí**.

El ataque Blue Mac Spoofing

El ataque Blue MAC Spoofing permite suplantar la identidad de un dispositivo de confianza y/o emparejado para atacar un teléfono móvil y utilizar sus credenciales para acceder a perfiles que requieren autorización y/o autenticación.

Se denomina Blue MAC Spoofing por su analogía con el clásico ataque MAC Spoofing en redes Ethernet, el cual permite a un atacante suplantar la dirección MAC de un equipo para suplantar su identidad y llevar a cabo acciones maliciosas contra el resto de equipos de la red, ya sea interceptando las comunicaciones dirigidas al equipo suplantado o utilizando sus credenciales con el fin de acceder a un sistema con acceso limitado.

Puesto que hay dos mecanismos de seguridad en Bluetooth, el ataque Blue MAC Spoofing se puede desarrollar en dos niveles:

- Suplantación de la dirección BD_ADDR de un dispositivo de confianza para acceder a perfiles que requieren autorización.

- Suplantación de la dirección BD_ADDR y obtención de la clave de enlace generada durante el emparejamiento para acceder a perfiles que requieren autenticación.

Alcance del ataque Blue Mac Spoofing

Para poder evaluar las consecuencias que tiene el hecho de desarrollar un ataque Blue MAC Spoofing con éxito sobre un teléfono móvil, es importante conocer los principales vectores de ataque en teléfonos Bluetooth.

Comandos AT

Los comandos AT son un conjunto de instrucciones codificadas que permiten configurar el teléfono móvil y enviarle órdenes a ejecutar. Básicamente, a través de los comandos AT un atacante puede realizar las siguientes acciones en un teléfono móvil:

- Realizar llamadas de voz y configurar desvíos de llamadas.

- Acceder a la agenda de contactos: leer, añadir, eliminar, …

- Acceder a la agenda de llamadas: últimas llamadas perdidas, recibidas y enviadas.

- Gestión de mensajes SMS: leer bandeja de entrada, escribir y enviar, eliminar, …

Acceso por OBEX

- A través de OBEX Object Push es posible el envío de archivos.

- A través de OBEX File Transfer es posible acceder al sistema de archivos del teléfono: subida, descarga, listado y borrado de archivos y directorios. Es posible acceder a archivos almacenados tanto en memoria del teléfono como en tarjetas extraíbles.

Veremos como la consecución con éxito del ataque Blue MAC Spoofing permite explotar ambos vectores de ataque.

Demostración Práctica del Ataque

Con el fin de demostrar el alcance del ataque Blue MAC Spoofing en un escenario habitual de interconexión de equipos Bluetooth. Este escenario permitirá observar lo fácil que resulta para un atacante suplantar la identidad de un dispositivo de confianza y/o emparejado con el fin de llevar a cabo ciertas acciones maliciosas contra un teléfono móvil.

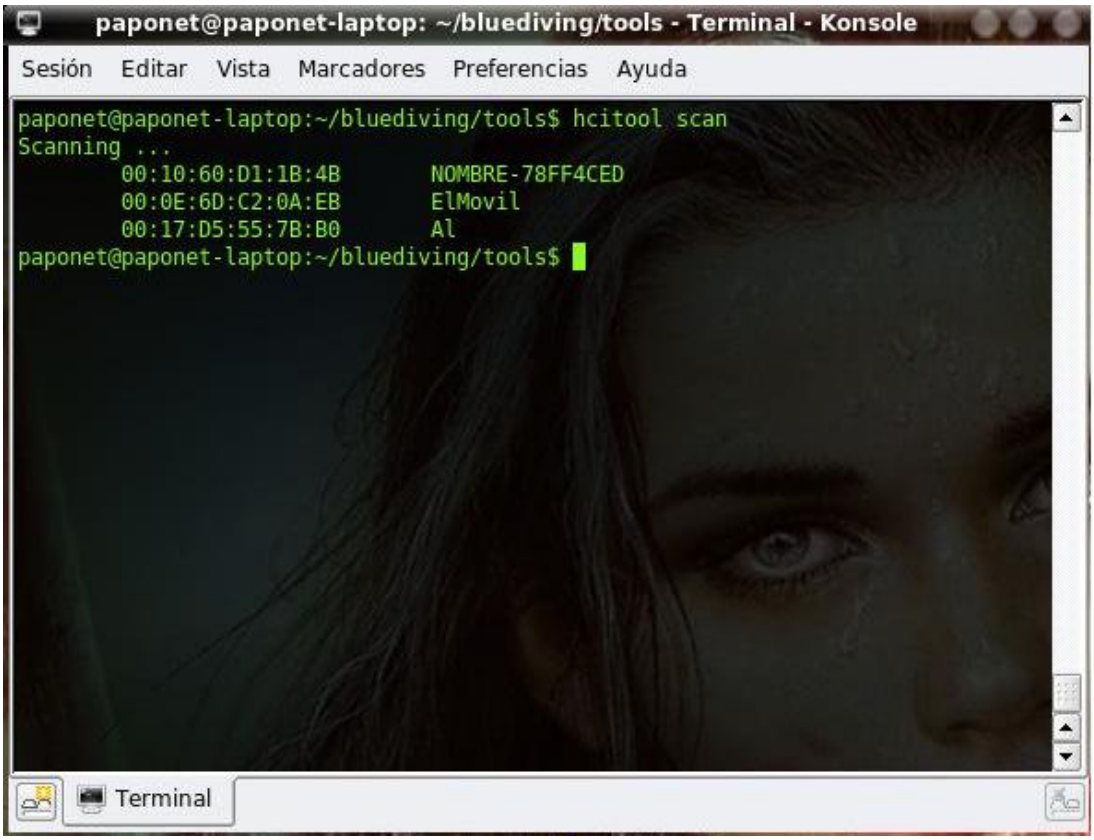
El escenario tiene como objetivo demostrar cómo es posible saltarse el mecanismo de autorización suplantando la identidad de un dispositivo de confianza.

En todos los escenarios, el equipo atacante será el mismo: un ordenador portátil con kubuntu y BlueZ, la pila de protocolos oficial para Linux.

Escenario 1

Un teléfono móvil se empareja con otro dispositivo móvil y marca a este último como confiable. El objetivo del atacante es suplantar la identidad del segundo teléfono y acceder al perfil de Carga de Objetos del teléfono, el cual requiere autorización, pero no autenticación, y permite el envío simple de archivos e intercambio de tarjetas de visita y contactos entre teléfonos.

Con el comando hcitool scan vemos todos los dispositivos bluetooth a nuestro alcance.

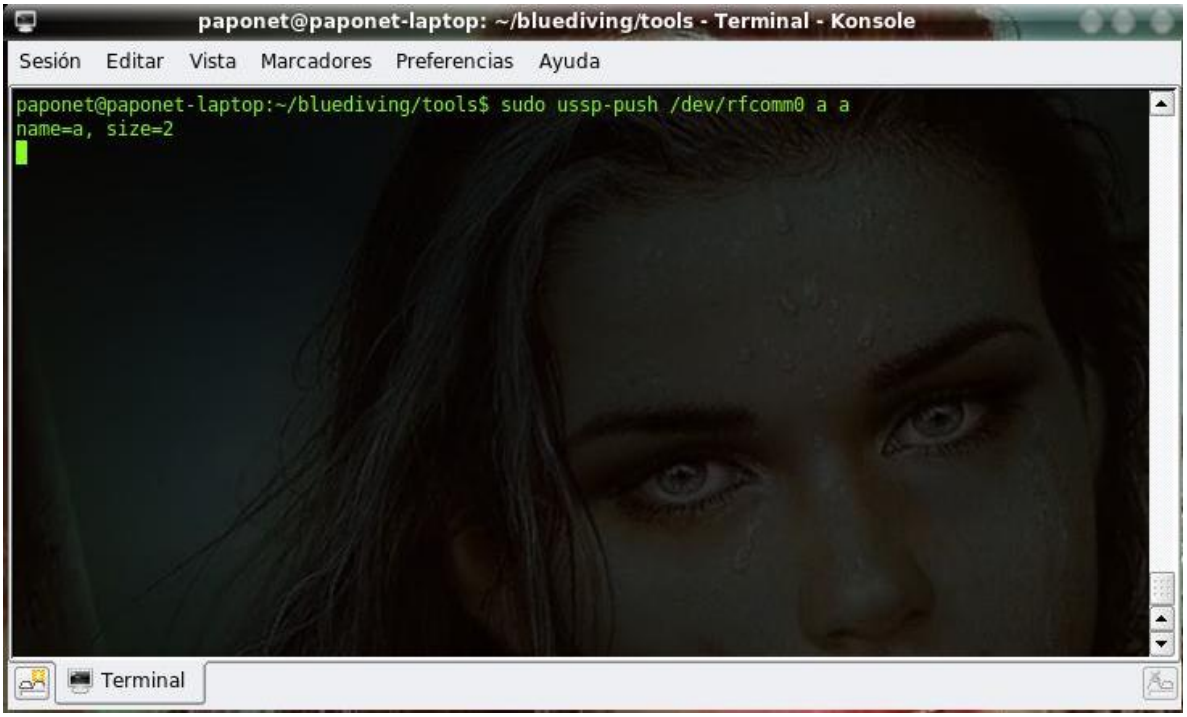


Saltándonos la autorización…

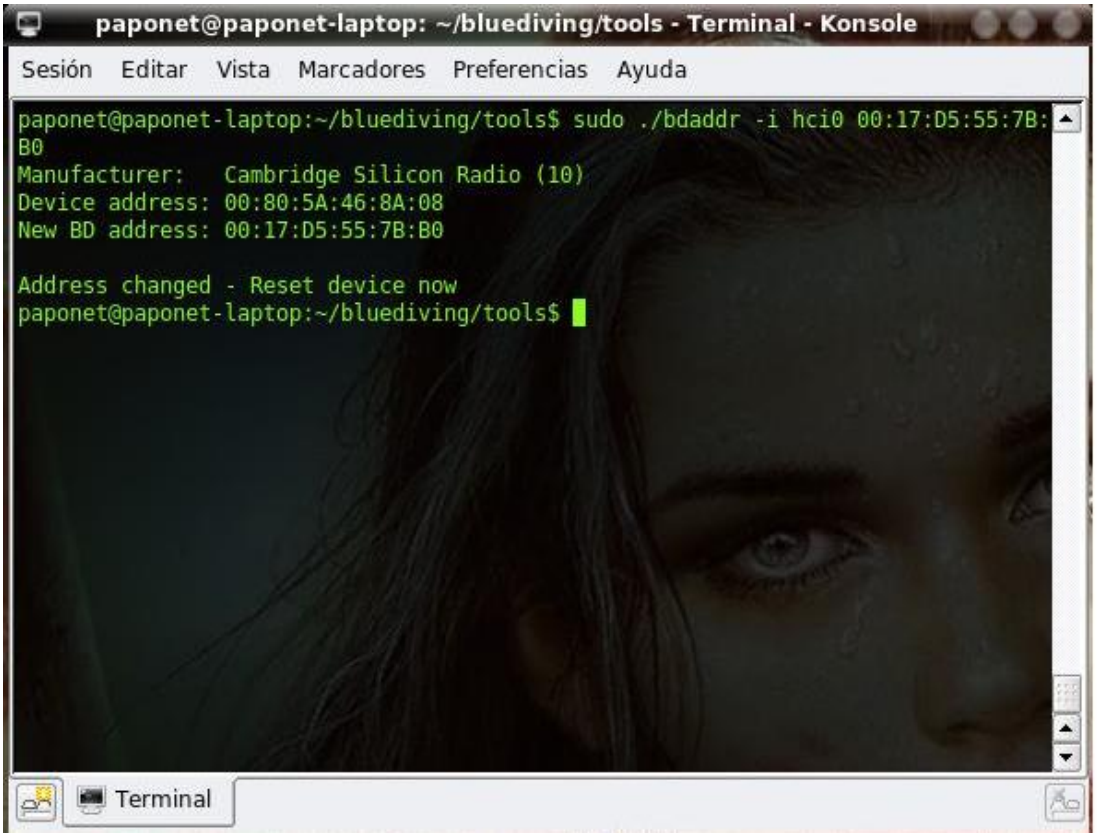
El objetivo del atacante es conectarse al Perfil de Carga de Objetos (OBEX Objet Push), disponible a través del canal 9, y enviar un archivo al teléfono.

Inicialmente, si el atacante intenta conectarse al Perfil de Carga de Objetos y enviar un archivo, al tratarse de un equipo no incluido en la lista de dispositivos de confianza, el usuario del primer teléfono deberá autorizar explícitamente la conexión.

Es muy probable que el usuario del primer teléfono no confíe en el emisor del envío y por ello deniegue el intento de conexión.

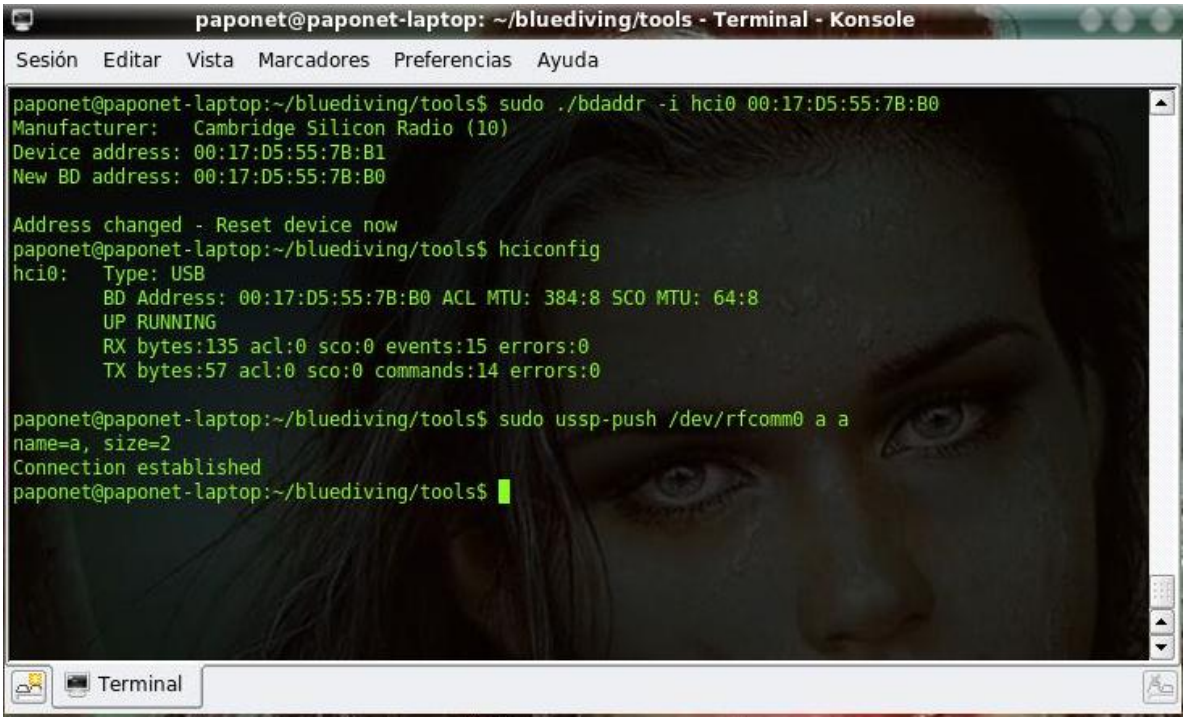


Para evitar el mecanismo de autorización, el atacante puede suplantar la identidad de un dispositivo de confianza del teléfono como, en este caso, otro teléfono que si que este autorizado. El atacante debe conocer la dirección BD_ADDR del equipo a suplantar y cambiar la dirección BD_ADDR del módulo Bluetooth utilizado por su equipo por esta nueva. Para ello, el atacante puede hacer uso de la herramienta bdaddr (Anexo1), que permite configurar la dirección BD_ADDR y otros parámetros en módulos hardware Bluetooth.



A partir de ese momento, el atacante está en disposición de conectarse a un perfil del teléfono móvil que requiera autorización de forma transparente ya que, a ojos del teléfono, el intento de conexión proviene de un dispositivo incluido en la lista de dispositivos de confianza.

Así pues, el atacante puede intentar conectarse de nuevo al Perfil de Carga de Objetos del teléfono y enviar el archivo. Esta vez, la conexión se realizará sin que el usuario del teléfono se percate de la acción.



El archivo enviado por el equipo atacante será automáticamente recibido por el teléfono móvil y almacenado en la bandeja de entrada de mensajes sin que el usuario haya tenido que autorizar la recepción del mismo.

Conclusiones

La conclusión principal es que la seguridad en los dispositivos bluetooth es bastante baja, ya que pueden tener acceso a tu teléfono en cuanto puedan suplantar la identidad de un dispositivo que tu tengas autorizado en tu teléfono, con las consecuencias que ello conlleva, te pueden leer los mensajes privados, números de teléfono, contraseñas incluso fotos que tengas almacenadas.

Por lo tanto se deduce del ataque Blue MAC Spoofing que si la seguridad de un equipo PC o PDA resulta comprometida, también puede verse comprometida la seguridad de aquellos teléfonos móviles Bluetooth emparejados con ese equipo.

Solución de Seguridad

En este documento se ha demostrado lo trivial que resulta suplantar la identidad de un dispositivo Bluetooth con el fin de acceder a otro utilizando sus credenciales. Esto se debe a que, en primera instancia, la arquitectura de seguridad en Bluetooth se definió a nivel de dispositivos, y no de usuarios. Dado que el ataque Blue MAC Spoofing explota un fallo en los mecanismos de seguridad utilizados por Bluetooth y que estos están definidos en la especificación de su protocolo, con el fin de solucionar esta vulnerabilidad sería preciso modificar la especificación del estándar.

En el caso de modificar la especificación de Bluetooth e implementar otros mecanismos de seguridad más robustos, los nuevos dispositivos fabricados serían incompatibles con los que actualmente se encuentran en el mercado, lo cual generaría un conflicto de interoperabilidad.

Se puede afirmar, por tanto, que no existe una solución a corto plazo y que, mientras tanto, todos los equipos fabricados que implementen tecnología Bluetooth seguirán siendo vulnerables a este ataque.

Mi consejo es no mantener activos en el teléfono aquellos enlaces con otros dispositivos Bluetooth emparejados más allá del tiempo necesario para su uso. En el caso de conexiones esporádicas o de baja periodicidad, mi consejo es realizar un emparejamiento de nuevo cada vez que se vaya a producir la comunicación y eliminar el enlace al final de la misma. En el caso de conexiones frecuentes con un mismo dispositivo, se puede mantener el enlace activo, siempre que el dispositivo esté bajo nuestro control o resulte difícil para un atacante extraer las claves de enlace de tal equipo, como, por ejemplo, un Manos Libres.

Palabras Clave

Bluetooth
Symbian
Nokia Conectivity Framework
Spoofing
Sniffer
Obex
Ethereal
EPOC
Carbide
Security Manager

Anexo 1 : Codigo bdaddr.c

```
/*
 *
 *  BlueZ - Bluetooth protocol stack for Linux
 *
 *        Copyright    (C)    2004-2006        Marcel    Holtmann
<marcel@holtmann.org>
 *
 *
 *  This program is free software; you can redistribute it and/or
modify
 *   it under the terms of the GNU General Public License as
published by
 *    the Free Software Foundation; either version 2 of the
License, or
 *  (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be
useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty
of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public
License
 *  along with this program; if not, write to the Free Software
 *   Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
02110-1301  USA
 *
 */

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <getopt.h>
#include <sys/ioctl.h>
#include <sys/socket.h>

#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>

#include "oui.h"

static int transient = 0;

static int generic_reset_device(int dd)
```

```
{
        bdaddr_t bdaddr;
        int err;

        err = hci_send_cmd(dd, 0x03, 0x0003, 0, NULL);
        if (err < 0)
                return err;

        return hci_read_bd_addr(dd, &bdaddr, 10000);
}

#define OCF_ERICSSON_WRITE_BD_ADDR    0x000d
typedef struct {
        bdaddr_t       bdaddr;
} __attribute__ ((packed)) ericsson_write_bd_addr_cp;
#define ERICSSON_WRITE_BD_ADDR_CP_SIZE 6

static int ericsson_write_bd_addr(int dd, bdaddr_t *bdaddr)
{
        struct hci_request rq;
        ericsson_write_bd_addr_cp cp;

        memset(&cp, 0, sizeof(cp));
        bacpy(&cp.bdaddr, bdaddr);

        memset(&rq, 0, sizeof(rq));
        rq.ogf    = OGF_VENDOR_CMD;
        rq.ocf    = OCF_ERICSSON_WRITE_BD_ADDR;
        rq.cparam = &cp;
        rq.clen   = ERICSSON_WRITE_BD_ADDR_CP_SIZE;
        rq.rparam = NULL;
        rq.rlen   = 0;

        if (hci_send_req(dd, &rq, 1000) < 0)
                return -1;

        return 0;
}

#define OCF_ERICSSON_STORE_IN_FLASH   0x0022
typedef struct {
        uint8_t        user_id;
        uint8_t        flash_length;
        uint8_t        flash_data[253];
} __attribute__ ((packed)) ericsson_store_in_flash_cp;
#define ERICSSON_STORE_IN_FLASH_CP_SIZE 255

static  int  ericsson_store_in_flash(int  dd,  uint8_t  user_id,
uint8_t flash_length, uint8_t *flash_data)
{
        struct hci_request rq;
        ericsson_store_in_flash_cp cp;

        memset(&cp, 0, sizeof(cp));
        cp.user_id = user_id;
        cp.flash_length = flash_length;
        if (flash_length > 0)
                memcpy(cp.flash_data, flash_data, flash_length);
```

```
        memset(&rq, 0, sizeof(rq));
        rq.ogf    = OGF_VENDOR_CMD;
        rq.ocf    = OCF_ERICSSON_STORE_IN_FLASH;
        rq.cparam = &cp;
        rq.clen   = ERICSSON_STORE_IN_FLASH_CP_SIZE;
        rq.rparam = NULL;
        rq.rlen   = 0;

        if (hci_send_req(dd, &rq, 1000) < 0)
                return -1;

        return 0;
}

static int csr_write_bd_addr(int dd, bdaddr_t *bdaddr)
{
        unsigned char cmd[] = { 0x02, 0x00, 0x0c, 0x00, 0x11,
0x47, 0x03, 0x70,
                                0x00, 0x00, 0x01, 0x00, 0x04, 0x00,
0x00, 0x00,
                                0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00 };

        unsigned char cp[254], rp[254];
        struct hci_request rq;

        if (transient)
                cmd[14] = 0x08;

        cmd[16] = bdaddr->b[2];
        cmd[17] = 0x00;
        cmd[18] = bdaddr->b[0];
        cmd[19] = bdaddr->b[1];
        cmd[20] = bdaddr->b[3];
        cmd[21] = 0x00;
        cmd[22] = bdaddr->b[4];
        cmd[23] = bdaddr->b[5];

        memset(&cp, 0, sizeof(cp));
        cp[0] = 0xc2;
        memcpy(cp + 1, cmd, sizeof(cmd));

        memset(&rq, 0, sizeof(rq));
        rq.ogf    = OGF_VENDOR_CMD;
        rq.ocf    = 0x00;
        rq.event  = EVT_VENDOR;
        rq.cparam = cp;
        rq.clen   = sizeof(cmd) + 1;
        rq.rparam = rp;
        rq.rlen   = sizeof(rp);

        if (hci_send_req(dd, &rq, 2000) < 0)
                return -1;

        if (rp[0] != 0xc2) {
                errno = EIO;
                return -1;
```

```
        }

        if ((rp[9] + (rp[10] << 8)) != 0) {
                errno = ENXIO;
                return -1;
        }

        return 0;
}

static int csr_reset_device(int dd)
{
        unsigned char cmd[] = { 0x02, 0x00, 0x09, 0x00,
                                0x00, 0x00, 0x01, 0x40, 0x00, 0x00,
                                0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00 };

        unsigned char cp[254], rp[254];
        struct hci_request rq;

        if (transient)
                cmd[6] = 0x02;

        memset(&cp, 0, sizeof(cp));
        cp[0] = 0xc2;
        memcpy(cp + 1, cmd, sizeof(cmd));

        memset(&rq, 0, sizeof(rq));
        rq.ogf    = OGF_VENDOR_CMD;
        rq.ocf    = 0x00;
        rq.event  = EVT_VENDOR;
        rq.cparam = cp;
        rq.clen   = sizeof(cmd) + 1;
        rq.rparam = rp;
        rq.rlen   = sizeof(rp);

        if (hci_send_req(dd, &rq, 2000) < 0)
                return -1;

        return 0;
}

#define OCF_TI_WRITE_BD_ADDR          0x0006
typedef struct {
        bdaddr_t       bdaddr;
} __attribute__ ((packed)) ti_write_bd_addr_cp;
#define TI_WRITE_BD_ADDR_CP_SIZE 6

static int ti_write_bd_addr(int dd, bdaddr_t *bdaddr)
{
        struct hci_request rq;
        ti_write_bd_addr_cp cp;

        memset(&cp, 0, sizeof(cp));
        bacpy(&cp.bdaddr, bdaddr);

        memset(&rq, 0, sizeof(rq));
        rq.ogf    = OGF_VENDOR_CMD;
```

```
        rq.ocf    = OCF_TI_WRITE_BD_ADDR;
        rq.cparam = &cp;
        rq.clen   = TI_WRITE_BD_ADDR_CP_SIZE;
        rq.rparam = NULL;
        rq.rlen   = 0;

        if (hci_send_req(dd, &rq, 1000) < 0)
                return -1;

        return 0;
}

#define OCF_BCM_WRITE_BD_ADDR         0x0001
typedef struct {
        bdaddr_t       bdaddr;
} __attribute__ ((packed)) bcm_write_bd_addr_cp;
#define BCM_WRITE_BD_ADDR_CP_SIZE 6

static int bcm_write_bd_addr(int dd, bdaddr_t *bdaddr)
{
        struct hci_request rq;
        bcm_write_bd_addr_cp cp;

        memset(&cp, 0, sizeof(cp));
        bacpy(&cp.bdaddr, bdaddr);

        memset(&rq, 0, sizeof(rq));
        rq.ogf    = OGF_VENDOR_CMD;
        rq.ocf    = OCF_BCM_WRITE_BD_ADDR;
        rq.cparam = &cp;
        rq.clen   = BCM_WRITE_BD_ADDR_CP_SIZE;
        rq.rparam = NULL;
        rq.rlen   = 0;

        if (hci_send_req(dd, &rq, 1000) < 0)
                return -1;

        return 0;
}

#define OCF_ZEEVO_WRITE_BD_ADDR               0x0001
typedef struct {
        bdaddr_t       bdaddr;
} __attribute__ ((packed)) zeevo_write_bd_addr_cp;
#define ZEEVO_WRITE_BD_ADDR_CP_SIZE 6

static int zeevo_write_bd_addr(int dd, bdaddr_t *bdaddr)
{
        struct hci_request rq;
        zeevo_write_bd_addr_cp cp;

        memset(&cp, 0, sizeof(cp));
        bacpy(&cp.bdaddr, bdaddr);

        memset(&rq, 0, sizeof(rq));
        rq.ogf    = OGF_VENDOR_CMD;
        rq.ocf    = OCF_ZEEVO_WRITE_BD_ADDR;
        rq.cparam = &cp;
```

```
        rq.clen   = ZEEVO_WRITE_BD_ADDR_CP_SIZE;
        rq.rparam = NULL;
        rq.rlen   = 0;

        if (hci_send_req(dd, &rq, 1000) < 0)
                return -1;

        return 0;
}

static int st_write_bd_addr(int dd, bdaddr_t *bdaddr)
{
        return  ericsson_store_in_flash(dd,  0xfe,  6,  (uint8_t  *)
bdaddr);
}

static struct {
        uint16_t compid;
        int (*write_bd_addr)(int dd, bdaddr_t *bdaddr);
        int (*reset_device)(int dd);
} vendor[] = {
        { 0,            ericsson_write_bd_addr,        NULL                  },
        { 10,           csr_write_bd_addr,     csr_reset_device       },
        { 13,           ti_write_bd_addr,      NULL                   },
        { 15,           bcm_write_bd_addr,     generic_reset_device   },
        { 18,           zeevo_write_bd_addr,   NULL                   },
        { 48,           st_write_bd_addr,      generic_reset_device   },
        { 57,           ericsson_write_bd_addr,
        generic_reset_device   },
        { 65535,        NULL,                  NULL                   },
};

static void usage(void)
{
        printf("bdaddr - Utility for changing the Bluetooth device
address\n\n");
        printf("Usage:\n"
                "\tbdaddr [-i <dev>] [-r] [-t] [new bdaddr]\n");
}

static struct option main_options[] = {
        { "device",    1, 0, 'i' },
        { "reset",     0, 0, 'r' },
        { "transient", 0, 0, 't' },
        { "help",      0, 0, 'h' },
        { 0, 0, 0, 0 }
};

int main(int argc, char *argv[])
{
        struct hci_dev_info di;
        struct hci_version ver;
        bdaddr_t bdaddr;
        char addr[18], oui[9], *comp;
        int i, dd, opt, dev = 0, reset = 0;

        bacpy(&bdaddr, BDADDR_ANY);
```

```
        while     ((opt=getopt_long(argc,     argv,     "+i:rth",
main_options, NULL)) != -1) {
                switch (opt) {
                case 'i':
                        dev = hci_devid(optarg);
                        if (dev < 0) {
                                perror("Invalid device");
                                exit(1);
                        }
                        break;

                case 'r':
                        reset = 1;
                        break;

                case 't':
                        transient = 1;
                        break;

                case 'h':
                default:
                        usage();
                        exit(0);
                }
        }

        argc -= optind;
        argv += optind;
        optind = 0;

        dd = hci_open_dev(dev);
        if (dd < 0) {
                fprintf(stderr,  "Can't  open  device  hci%d:  %s
(%d)\n",
                                                dev,
strerror(errno), errno);
                exit(1);
        }

        if (hci_devinfo(dev, &di) < 0) {
                fprintf(stderr, "Can't get device info for hci%d:
%s (%d)\n",
                                                dev,
strerror(errno), errno);
                hci_close_dev(dd);
                exit(1);
        }

        if (hci_read_local_version(dd, &ver, 1000) < 0) {
                fprintf(stderr,  "Can't  read  version  info  for
hci%d: %s (%d)\n",
                                                dev,
strerror(errno), errno);
                hci_close_dev(dd);
                exit(1);
        }

        if (!bacmp(&di.bdaddr, BDADDR_ANY)) {
```

```
                if (hci_read_bd_addr(dd, &bdaddr, 1000) < 0) {
                        fprintf(stderr,  "Can't  read  address  for
hci%d: %s (%d)\n",
                                                        dev,
strerror(errno), errno);
                        hci_close_dev(dd);
                        exit(1);
                }
        } else
                bacpy(&bdaddr, &di.bdaddr);

        printf("Manufacturer:   %s (%d)\n",
                        bt_compidtostr(ver.manufacturer),
ver.manufacturer);

        ba2oui(&bdaddr, oui);
        comp = ouitocomp(oui);

        ba2str(&bdaddr, addr);
        printf("Device address: %s", addr);

        if (comp) {
                printf(" (%s)\n", comp);
                free(comp);
        } else
                printf("\n");

        if (argc < 1) {
                hci_close_dev(dd);
                exit(0);
        }

        str2ba(argv[0], &bdaddr);
        if (!bacmp(&bdaddr, BDADDR_ANY)) {
                hci_close_dev(dd);
                exit(0);
        }

        for (i = 0; vendor[i].compid != 65535; i++)
                if (ver.manufacturer == vendor[i].compid) {
                        ba2oui(&bdaddr, oui);
                        comp = ouitocomp(oui);

                        ba2str(&bdaddr, addr);
                        printf("New BD address: %s", addr);

                        if (comp) {
                                printf(" (%s)\n\n", comp);
                                free(comp);
                        } else
                                printf("\n\n");


                        if (vendor[i].write_bd_addr(dd, &bdaddr) <
0) {
                                fprintf(stderr,  "Can't  write  new
address\n");
                                hci_close_dev(dd);
```

```
                                exit(1);
                        }

                        printf("Address changed - ");

                        if (reset && vendor[i].reset_device) {
                                if (vendor[i].reset_device(dd) < 0)
{
                                        printf("Reset         device
manually\n");
                                } else {
                                        ioctl(dd,       HCIDEVRESET,
dev);
                                        printf("Device         reset
successully\n");
                                }
                        } else {
                                printf("Reset device now\n");
                        }

                        //ioctl(dd, HCIDEVRESET, dev);
                        //ioctl(dd, HCIDEVDOWN, dev);
                        //ioctl(dd, HCIDEVUP, dev);

                        hci_close_dev(dd);
                        exit(0);
                }

        hci_close_dev(dd);

        printf("\n");
        fprintf(stderr, "Unsupported manufacturer\n");

        exit(1);
}
```

BIBLIOGRAFÍA

LEIGH EDWARDS, RICHARD BARKER, DEVELOPING SERIES 60 APPLICATIONS: A GUIDE FOR SYMBIAN OS C++. ADDISON WESLEY MARCH 01, 2004

PROGRAMMING PLATFORM AND FOR THE SERIES 60 SYMBIAN OS, DIGIA INC. HELSINKI, FINLAND

STEVE BABIN, DEVELOPING SOFTWARE FOR SYMBIAN OS - AN INTRODUCTION TO CREATING SMARTPHONE APPLICATIONS IN C++, JOHN WILEY & SONS, LTD

RICHARD HARRISON, SYMBIAN OS C++ FOR MOBILE PHONES VOLUME 2 - ISBN10-0470871083.

JO STICHBURY, SYMBIAN OS EXPLAINED, EFFECTIVE C++ PROGRAMMING FOR SMARTPHONES

BLOG DE GOSPEL [ALBERTO MORENO TABLADO], INGENIERO I+D DE SEGURIDAD MOBILE. HTTP://ELBLOGDEGOSPEL.BLOGSPOT.COM/

JAIME TEJEDA DE HOYOS, JOSÉ MARÍA SIERRA CÁMARA, SEGURIDAD EN SYMBIAN: ENTORNO DE DETERMINACIÓN DE LA INTEGRIDAD ADAPTABLE (EDIA) (POYECTO FIN DE CARRERA DEPARTAMENTO DE INFORMÁTICA UNIVERSIDAD CARLOS III DE MADRID)

ÁNGEL GARCÍA MORENO, ANTONIO IZQUIERDO MANZANARES, ANÁLISIS DE SEGURIDAD EN DISPOSITIVOS MÓVILES CON SISTEMA SYMBIAN.(PROYECTO FIN DE CARRERA UNIVERSIDAD CARLOS III DE MADRID)

SNIFFER_UCM HERRAMIENTA DE MONITOREO DE TRÁFICO, SISTEMAS INFORMATICOS UCM 2003-2004

AUTORIZACIÓN

Alberto Macho González y Francisco Javier Pérez Escrivá autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales, tanto la propia memoria, como el código y/o la documentación.

Fdo Fdo

Alberto Macho González Fco Javier Pérez Escrivá